If you see a formula in the Physical Review that extends over a quarter of a page, forget it. It's wrong. Nature isn't that complicated . Bernd T Matthias.

Artificial Biochemistry

Monopolin Circuits

Luca Cardelli

Microsoft Research

The Microsoft Research - University of Trento
Centre for Computational and Systems Biology

Trento, 2006-05-22..26

www.luca.demon.co.uk/ArtificialBiochemistry.htm

Polin Diagrams

Influence Diagrams

1506 26 NOVEMBER 2004 VOL 306 SCIENCE www.sciencemag.org

CELL SIGNALING

VIEWPOINT

Common and Distinct Elements in Cellular Signaling via EGF and FGF Receptors

Joseph Schlessinger*

What do they mean?
Usually NOTHING.

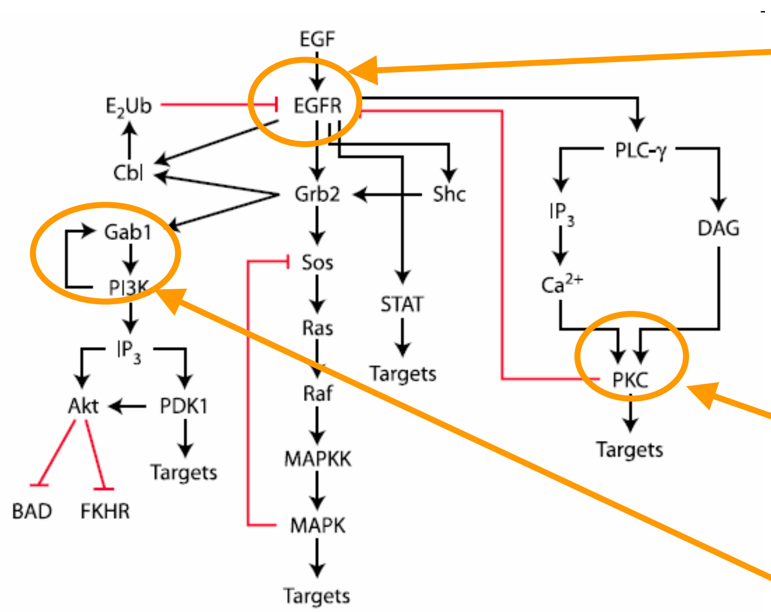


Fig. 1. Cell signaling by EGF or FGF receptors. An abbreviated version of signaling by EGFR (left) and FGFR (right). Detailed description is presented in STKE Connections Maps (9, 10). Stimulatory and inhibitory stimuli are depicted in black and red, respectively. Abbreviations: HSPG,

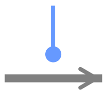
Is EGFR regulated, shut down, or oscillating, by the negative feedback loop?

Is this an AND or an OR?

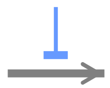
How can this positive feedback loop ever reset once started?

Influence Automata

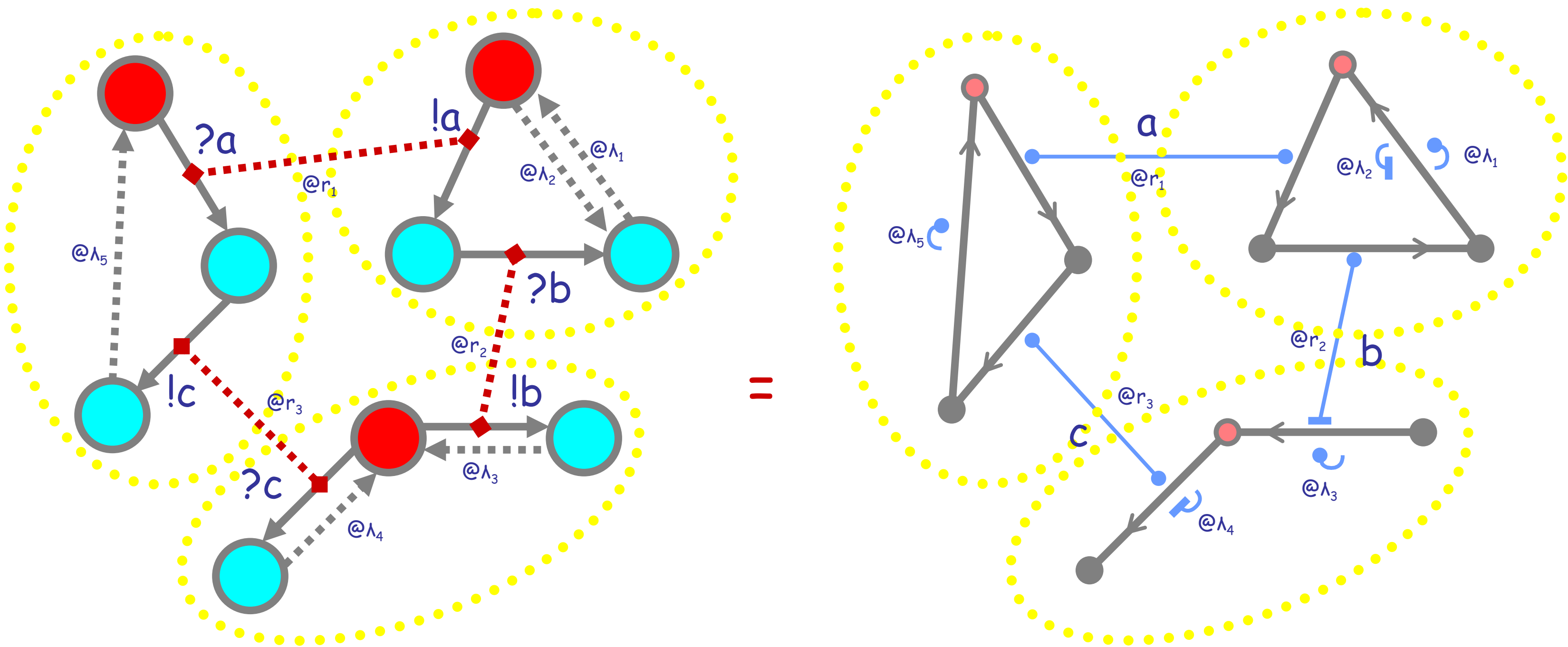
Nonetheless, the basic idea of influence diagrams can be cast as an alternative notation for automata.



positive influence on transition

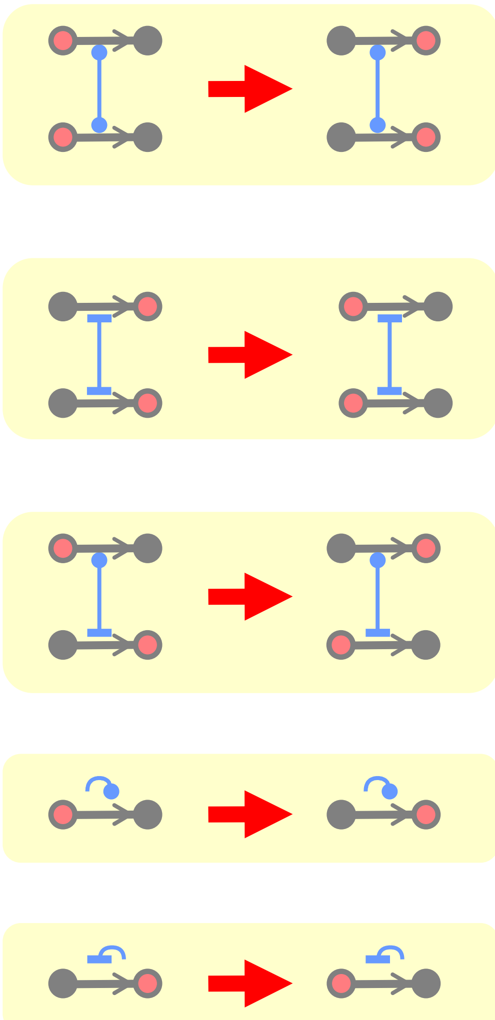


negative influence on transition

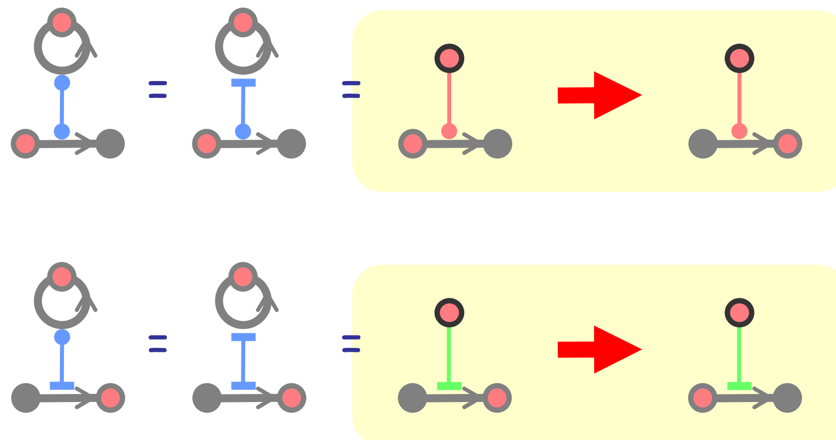


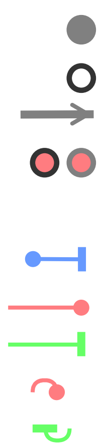
Influence Transitions

5 basic influence transitions



Plus 2 abbreviations for self-loops





node
pole (self-loop)
stem (unique arc between two nodes or poles)
current node or pole (unique in each automaton)

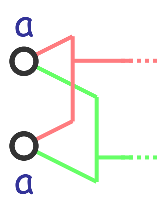
Influence between stems
Excitation between pole and stem
Inhibition between pole and stem
Accretion on a stem
Degradation on a stem

Influence Diagram Conventions

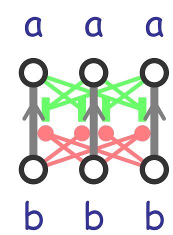
Influence diagrams where the only two-ended influences are excitation and inhibition between poles and stems, are called POLIN DIAGRAMS.

It is convenient to name poles: those names correspond to the channel names in automata. (It does not seem critical to name other nodes.)

By convention, then, equally named poles are always equally connected (otherwise they would not correspond to channel names).

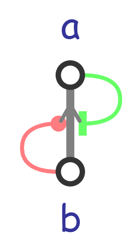


By definition, each two-ended influence connects separate automata.



A population of 3 automata

But we often represent population schemas by two-ended influences within the same diagram:



A population schema for a population of size n of such automata

Still, a two-ended influence is always intended between two separate automata.

Polin Diagrams to Communicating Automata

- Each node in a polin becomes a node in an automaton.

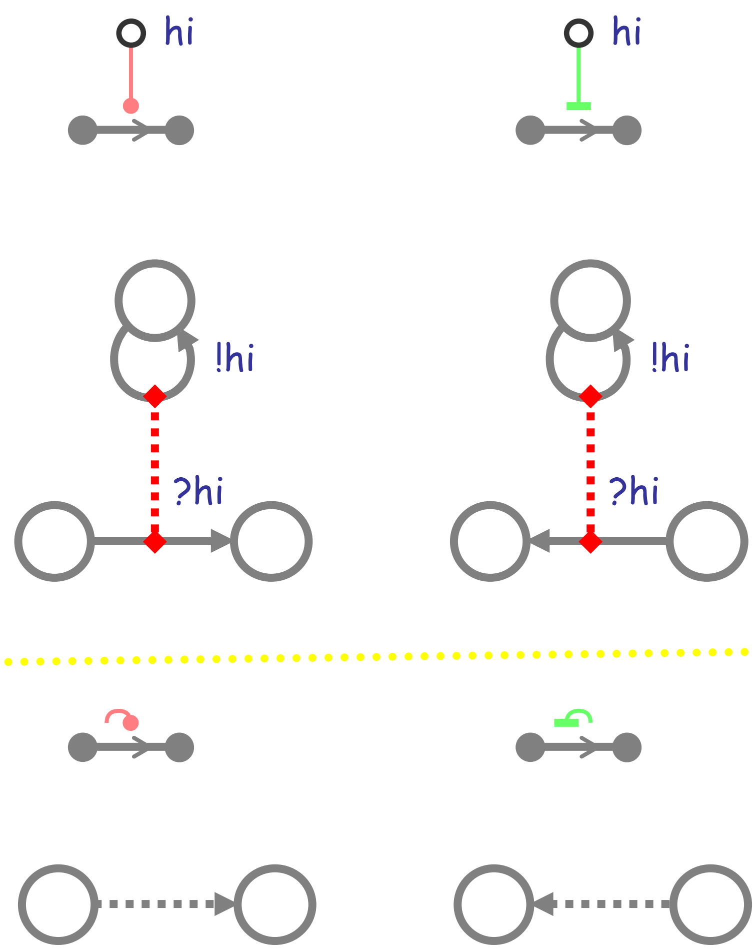
- Each pole becomes an output self-transition in the automaton, with the same name.

- Each pole-to-stem connection becomes an input transition in the automaton between the stem nodes (reverse transition if inhibition). The name of the transition comes from the name of the source pole.

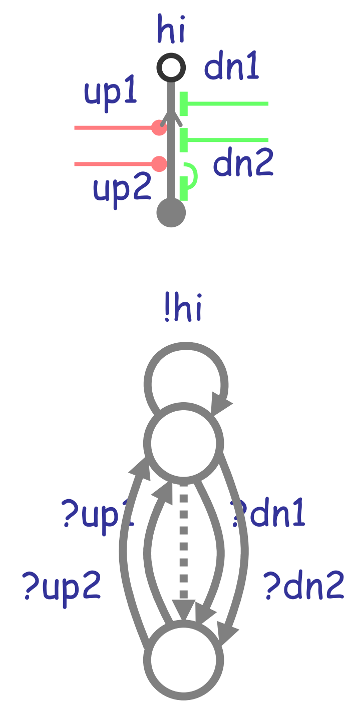
- Accretion/degradation arcs, become delays in the appropriate direction.

- Multiple connections on a single stem become multiple transitions between nodes.

Interactions



Multiple controls

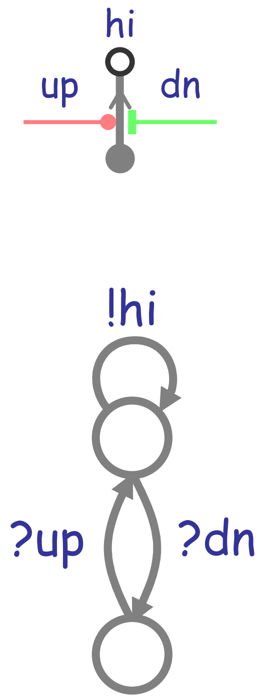


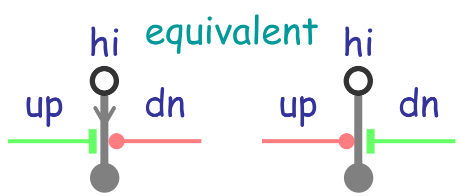
Monopolins

Monopolins

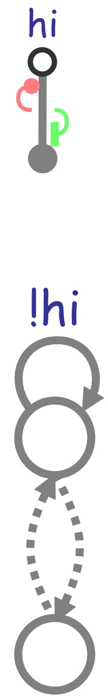
Constant

Excitation/Inhibition





Accretion/Degradation




```
Mon() = !hi; Mon()
```

```
Mon_hi(hi,up,dn) =
    !hi; Mon_hi(hi,up,dn)
    + ?dn; Mon_lo(hi,up,dn)

Mon_lo(hi,up,dn) =
    ?up; Mon_hi(hi,up,dn)
```

A monopolin may have one or more poles, but all such poles are named with a single name. Other nodes are unnamed.

Nodes are connected by *oriented stems*.

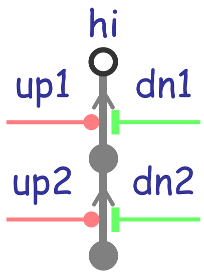
Activation and inhibition *arcs* connect poles to stems.

The orientation of a stem can be omitted when clear by convention (activation is then always towards a pole, and inhibition away from it).

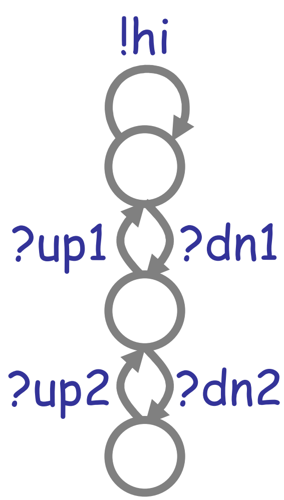
One node can be marked as current (red) to indicate the current state of a specific polin instance.

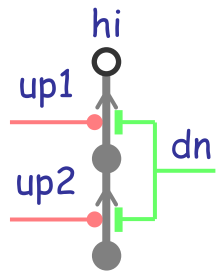
Names that may appear on arcs do not belong to the arcs: they simply indicate that the arc comes from some pole with that name.

More Monopolins

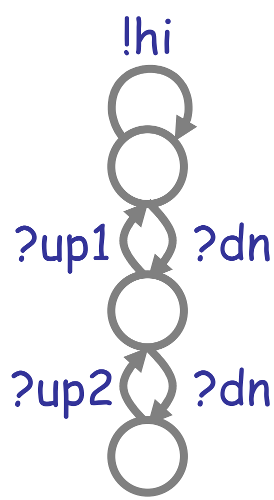


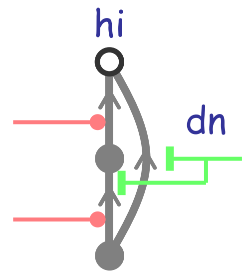
“and-up/and-dn”



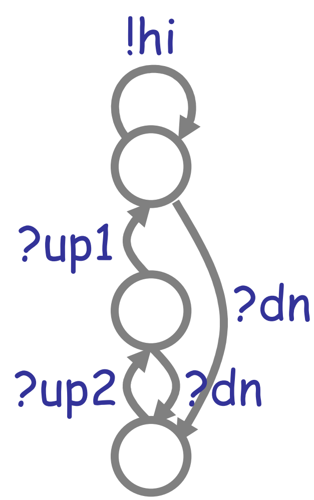


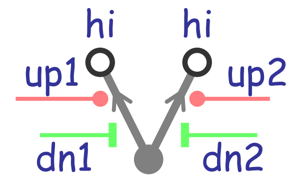
“and-up/or-dn”



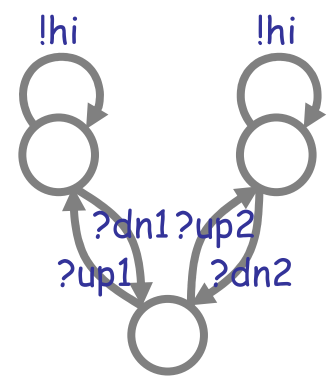


“and-up/or-dn”



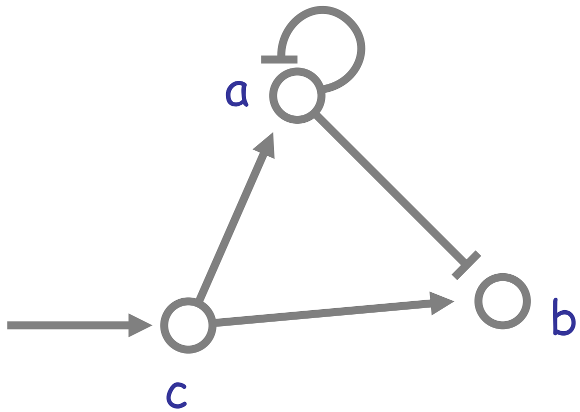


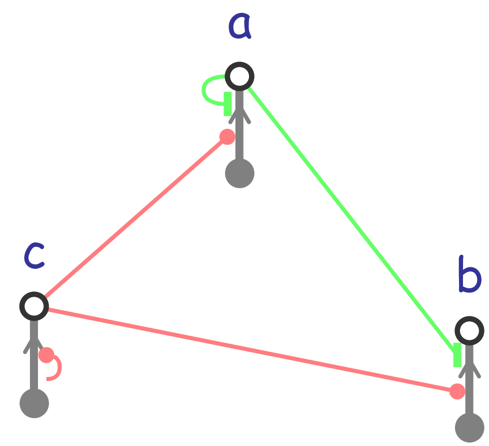
two copies of the “same” pole
(poles with the same name must have exactly the same outgoing arcs)



(each of the !hi outputs obviously connects to all the ?hi transitions anywhere in the network)

Influence Diagrams by Monopolins





This simple increment/decrement idea can actually give good results, if done carefully:

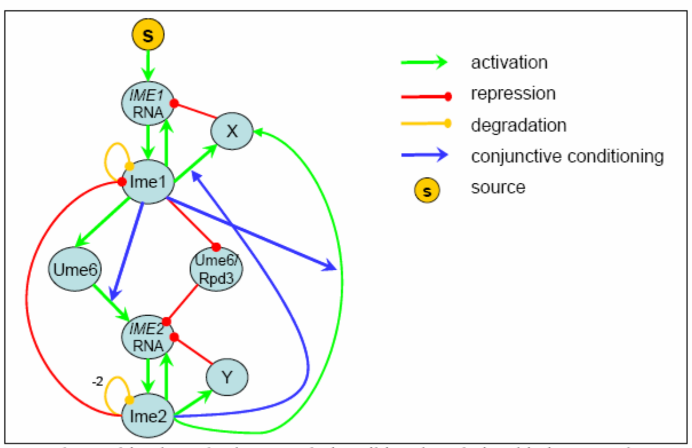


Fig. 2. The working hypothesis network describing the relationship between the expression of *IME1* and *IME2*.

Faithful Modeling of Transient Behavior in Developmental Pathways

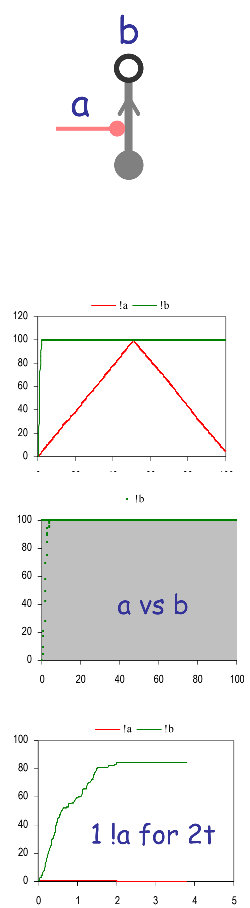
Amir Rubinstein[1], Vyacheslav Gurevich[2], Yona Kassir[2] and Ron Y. Pinter[1]

[1]Dept. of Computer Science, Technion – Israel Institute of Technology, Haifa 32000, Israel
[2]Dept. of Biology, Technion – Israel Institute of Technology, Haifa 32000, Israel

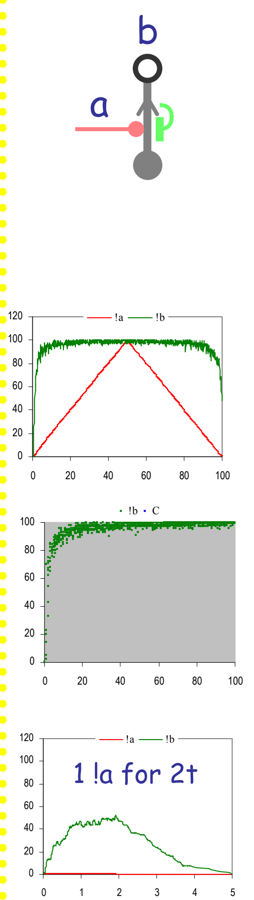
But CAVEAT EMPTOR:
influence diagrams in biology are not meant to convey semantics!

Amplifiers

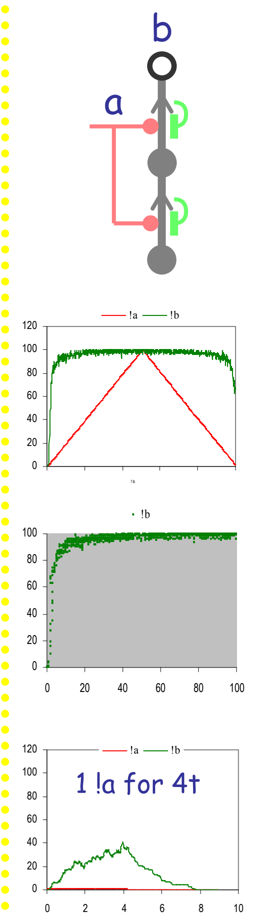
Amplifiers



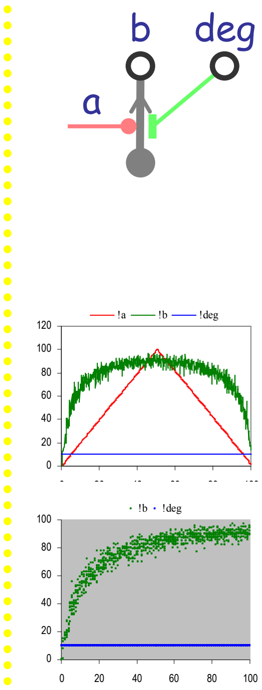

```
directive sample 5.0 1000
directive plot !a; !b

new a@1.0:chan new b@1.0:chan

let Amp_hi(a:chan, b:chan) =
  !b; Amp_hi(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; Amp_hi(a,b)

run 100 of Amp_lo(a,b)

new tick:chan
let clock(t:float) =       (* sends a tick every t time *)
  (val ti = t/100.0 val d = 1.0/ti
   let step(n:int) = if n<=0 then !tick; clock(t) else
delay@d; step(n-1)
   run step(100))
let S() = do !a; S() or ?tick; ()
run 1 of (clock(2.0) | S())
```




```
directive sample 5.0 1000
directive plot !a; !b

new a@1.0:chan new b@1.0:chan

let Amp_hi(a:chan, b:chan) =
  do !b; Amp_hi(a,b) or delay@1.0; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; Amp_hi(a,b)

run 100 of Amp_lo(a,b)

new tick:chan
let clock(t:float) =       (* sends a tick every t time *)
  (val ti = t/100.0 val d = 1.0/ti
   let step(n:int) = if n<=0 then !tick; clock(t) else delay@d;
step(n-1)
   run step(100))
let S() = do !a; S() or ?tick; ()
run 1 of (clock(2.0) | S())
```

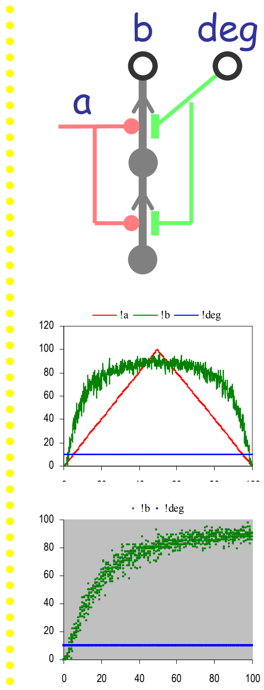



```
directive sample 10.0 1000
directive plot !a; !b

new a@1.0:chan new b@1.0:chan
val del = 1.0

let Amp_hi(a:chan, b:chan) =
  do !b; Amp_hi(a,b) or delay@del; Amp_mi(a,b)
and Amp_mi(a:chan, b:chan) =
  do ?a; Amp_hi(a,b) or delay@del; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; Amp_mi(a,b)

run 100 of Amp_lo(a,b)

new tick:chan
let clock(t:float) =       (* sends a tick every t time *)
  (val ti = t/100.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t) else delay@d; step(n-1)
   run step(100))

let S() = do !a; S() or ?tick; ()

run 10 of (clock(4.0) | S())
```



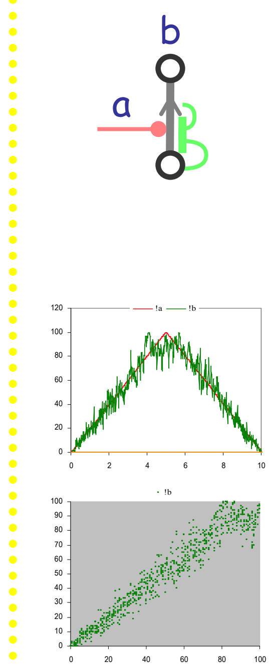
100*b with n*a vs 10*deg

```
directive sample 100.0 1000
directive plot !a; !b; !deg

new a@1.0:chan new b@1.0:chan new deg@1.0:chan
new _up:chan new _dn:chan
val del = 1.0

let Mon_hi(hi:chan, up:chan, dn:chan) =
  do !hi; Mon_hi(hi,up,dn) or ?dn; Mon_lo(hi,up,dn)
and Mon_lo(hi:chan, up:chan, dn:chan) =
  ?up; Mon_hi(hi,up,dn)

run (100 of Mon_lo(b,a,deg) | 10 of Mon_hi(deg,_up,_dn))

new tick:chan
let clock(t:float) =       (* sends a tick every t time *)
  (val ti = t/100.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t) else delay@d; step(n-1)
   run step(100))

let S() = do !a; S() or ?tick; (S() | S())

run 1 of (clock(3.0) | S())
```



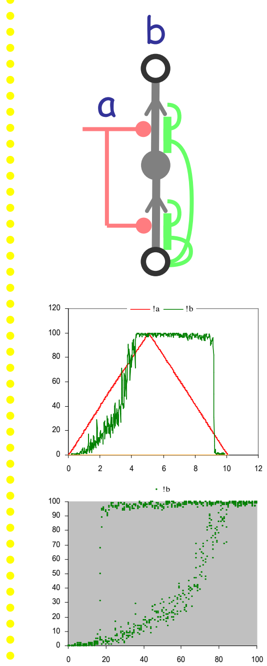
100*b with n*a vs 10*deg

```
directive sample 100.0 1000
directive plot !a; !b; !c; !deg

new a@1.0:chan new b@1.0:chan new c@1.0:chan new
deg@1.0:chan
new _up:chan new _dn:chan

let Mon_hi(hi:chan, up:chan, dn:chan) =
  do !hi; Mon_hi(hi,up,dn) or ?dn; Mon_mi(hi,up,dn)
and Mon_mi(hi:chan, up:chan, dn:chan) =
  do ?up; Mon_hi(hi,up,dn) or ?dn; Mon_lo(hi,up,dn)
and Mon_lo(hi:chan, up:chan, dn:chan) =
  ?up; Mon_mi(hi,up,dn)

(* run (100 of Mon_lo(b,a,deg) | 10 of Mon_hi(deg,_up,_dn))
*)
run (100 of (Mon_lo(b,a,deg) | Mon_lo(c,b,deg)) | 20 of
Mon_hi(deg,_up,_dn))

let clock(tick:chan, t:float) =       (* sends a tick every t time
*)
  (val ti = t/100.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(tick,t) else delay@d; step(n-1)
   run step(100))

new tick:chan
let S() = do !a; S() or ?tick; (S() | S())
run 1 of (clock(tick,3.0) | S())
```



linear

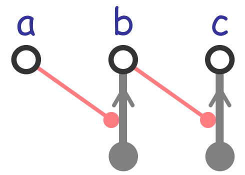
```
directive sample 100.0 1000
directive plot !a; !b; !r

new a@1.0:chan new b@1.0:chan new r@1.0:chan

let Amp_lo(a:chan, r:chan, b:chan) =
  do !r; Amp_lo(a,r,b)
  or ?a; Amp_hi(a,r,b)
and Amp_hi(a:chan, r:chan, b:chan) =
  do !b; Amp_hi(a,r,b)
  or ?r; Amp_lo(a,r,b)
  or delay@1.0; Amp_lo(a,r,b)

run 100 of Amp_lo(a,r,b)

let clock(t:float, tick:chan) =       (* sends a tick every t time
*)
  (val ti = t/100.0 val d = 1.0/ti     (* by 100-step erlang
timers *)
   let step(n:int) = if n<=0 then !tick; clock(t,tick) else
delay@d; step(n-1)
   run step(100))
let S1(a:chan, tock:chan) =  do !a; S1(a,tock) or ?tock; ()
let SN(n:int, t:float, a:chan, tick:chan, tock:chan) =
  if n=0 then clock(t, tock) else ?tick; (S1(a,tock) | SN(n-
1,t,a,tick,tock))
let raisingfalling(a:chan, n:int, t:float) =
  (new tick:chan new tock:chan
   run (clock(t,tick) | SN(n,t,a,tick,tock)))

run raisingfalling(a,100,0.5)
```

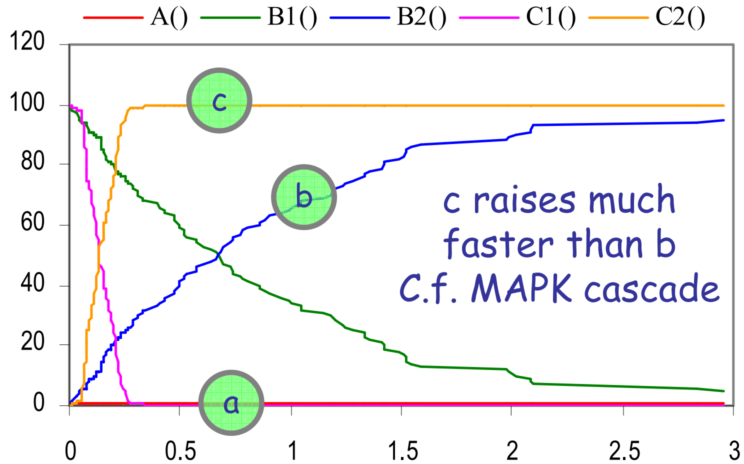


hysteresis

```
directive sample 100.0 1000
directive plot !a; !b; !r

new a@1.0:chan new b@1.0:chan new r@1.0:chan

let Amp2_lo(a:chan, r:chan, b:chan) =
  do !r; Amp2_lo(a,r,b)
  or ?a; Amp2_mi(a,r,b)
and Amp2_mi(a:chan, r:chan, b:chan) =
  do ?a; Amp2_hi(a,r,b)
  or ?r; Amp2_lo(a,r,b)
  or delay@1.0; Amp2_lo(a,r,b)
and Amp2_hi(a:chan, r:chan, b:chan) =
  do !b; Amp2_hi(a,r,b)
  or ?r; Amp2_lo(a,r,b)
  or delay@1.0; Amp2_mi(a,r,b)

run 100 of Amp2_lo(a,r,b)

let clock(t:float, tick:chan) =       (* sends a tick every t
time *)
  (val ti = t/100.0 val d = 1.0/ti     (* by 100-step erlang
timers *)
   let step(n:int) = if n<=0 then !tick; clock(t,tick) else
delay@d; step(n-1)
   run step(100))
let S1(a:chan, tock:chan) =  do !a; S1(a,tock) or ?tock; ()
let SN(n:int, t:float, a:chan, tick:chan, tock:chan) =
  if n=0 then clock(t, tock) else ?tick; (S1(a,tock) | SN(n-
1,t,a,tick,tock))
let raisingfalling(a:chan, n:int, t:float) =
  (new tick:chan new tock:chan
   run (clock(t,tick) | SN(n,t,a,tick,tock)))

run raisingfalling(a,100,0.5)
```

Basic Excitation Cascade






```
directive sample 3.0 10000
directive plot A(); B1(); B2(); C1(); C2()

new a@1.0:chan()   new b@1.0:chan()   new c@1.0:chan()

let A() = !a; A()
let B1() = ?a; B2()   and B2() = !b; B2()
let C1() = ?b; C2()   and C2() = !c; C2()

run 1 of A()   run 100 of B1()  run 100 of C1()
```

Abstracting a little library of composable monopolin components

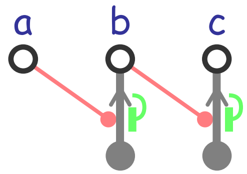
$Amp_{hi}(hi,up,dn) =$
  $!hi;\ Amp_{hi}(hi,up,dn)$
  $+\ ?dn;\ Amp_{lo}(hi,up,dn)$

$Amp_{lo}(hi,up,dn) =$
  $?up;\ Amp_{hi}(hi,up,dn)$

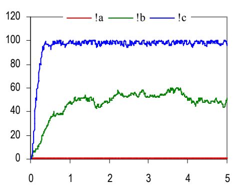
$Amp_{hi}(a,-,-)\ |$
$100\ of\ Amp_{lo}(b,a,-)\ |$
$100\ of\ Amp_{lo}(c,b,-)$

```
directive sample 1.0 10000
directive plot !a; !b; !c

type A = chan (* action *)
type S = chan (* state *)

let Amp_hi(hi:S, up:A, dn:A) =
  do !hi; Amp_hi(hi,up,dn) or ?dn; Amp_lo(hi,up,dn)
and Amp_lo(hi:S, up:A, dn:A) =
  ?up; Amp_hi(hi,up,dn)

new _up:chan new _dn:chan (*unused wiring *)
new a@1.0:chan new b@1.0:chan new c@1.0:chan

let A_hi() = Amp_hi(a,_up,_dn)
let B_lo() = Amp_lo(b,a,_dn)
let C_lo() = Amp_lo(c,b,_dn)

run 1 of A_hi()  run 100 of B_lo()  run 100 of C_lo()
```

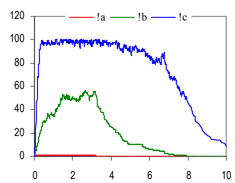
Excitation Cascade with Decay



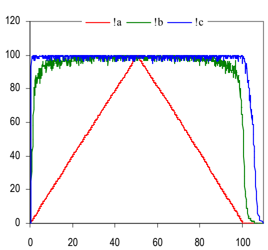
When competing with degradation, the a signal (very weak) is not able to fully raise b. However, c is still raised.

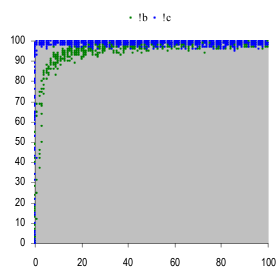


1 !a



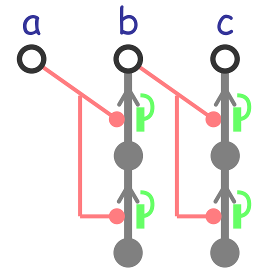
1 !a for 3t



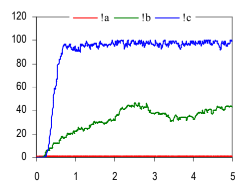


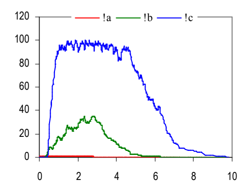
```
directive sample 5.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let Amp_hi(a:chan, b:chan) =
  do !b; Amp_hi(a,b) or delay@del; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; Amp_hi(a,b)

run (replicate !a | 100 of (Amp_lo(a,b) | Amp_lo(b,c)))
```

```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let Amp_hi(a:chan, b:chan) =
  do !b; Amp_hi(a,b) or delay@del; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; Amp_hi(a,b)

run 100 of (Amp_lo(a,b) | Amp_lo(b,c))

new tick:chan
let clock(t:float) =      (* sends a tick every t time *)
  (val ti = t/100.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t) else delay@d; step(n-1)
   run step(100))

let S() = do !a; S() or ?tick; ()
run 1 of (clock(3.0) | S())
```

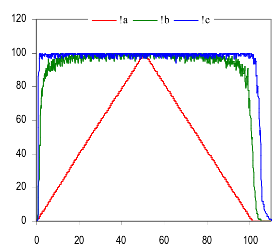


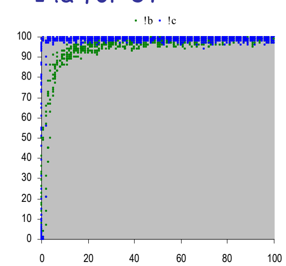
Double excitation seems to make the off response a bit quicker.





1 !a for 3t

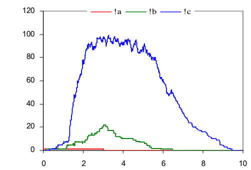




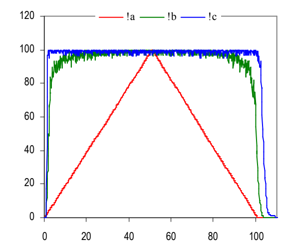
```
directive sample 5.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let Amp_hi(a:chan, b:chan) =
  do !b; Amp_hi(a,b) or delay@del; Amp_mi(a,b)
and Amp_mi(a:chan, b:chan) =
  do ?a; Amp_hi(a,b) or delay@del; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; Amp_mi(a,b)

run (replicate !a | 100 of (Amp_lo(a,b) | Amp_lo(b,c)))
```

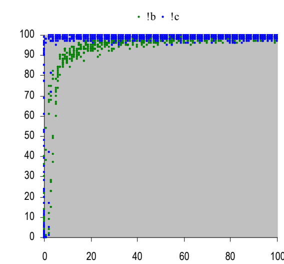
```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let Amp_hi(a:chan, b:chan) =
  do !b; Amp_hi(a,b) or delay@del; Amp_mi(a,b)
and Amp_mi(a:chan, b:chan) =
  do ?a; Amp_hi(a,b) or delay@del; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; Amp_mi(a,b)

run 100 of (Amp_lo(a,b) | Amp_lo(b,c))

new tick:chan
let clock(t:float) =      (* sends a tick every t time *)
  (val ti = t/100.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t) else delay@d; step(n-1)
   run step(100))

let S() = do !a; S() or ?tick; ()
run 1 of (clock(3.0) | S())
```

Triple excitation



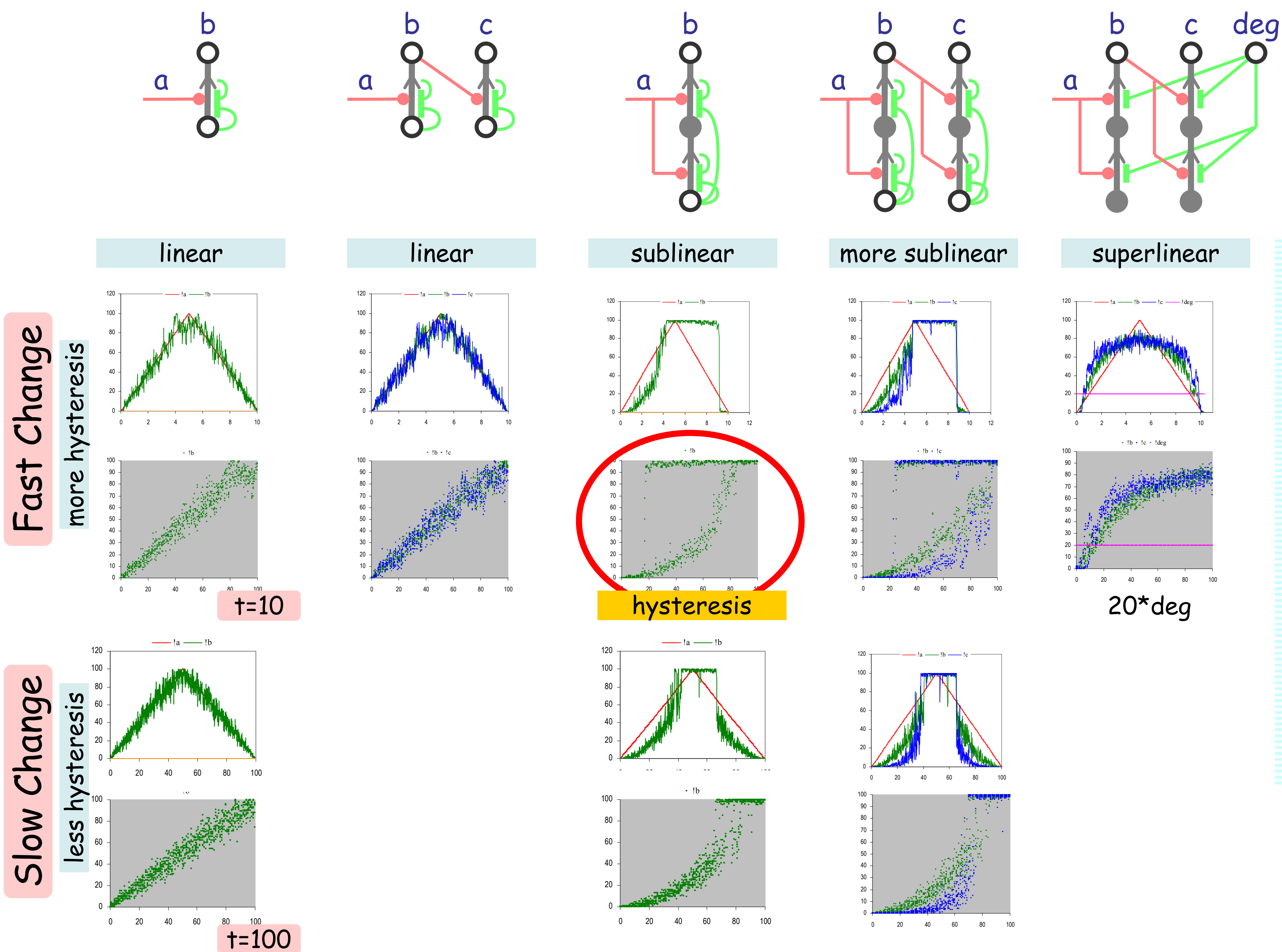
1 !a for 3t





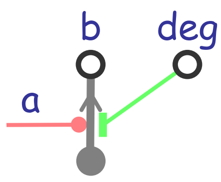
```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let Amp_hi(a:chan, b:chan) =
  do !b; Amp_hi(a,b) or delay@del; Amp_mi1(a,b)
and Amp_mi1(a:chan, b:chan) =
  do ?a; Amp_hi(a,b) or delay@del; Amp_mi2(a,b)
and Amp_mi2(a:chan, b:chan) =
  do ?a; Amp_mi1(a,b) or delay@del; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; Amp_mi2(a,b)

run 100 of (Amp_lo(a,b) | Amp_lo(b,c))

new tick:chan
let clock(t:float) =      (* sends a tick every t time *)
  (val ti = t/100.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t) else delay@d; step(n-1)
   run step(100))

let S() = do !a; S() or ?tick; ()

run 1 of (clock(3.0) | S())
```

Double Excitation and Hysteresis



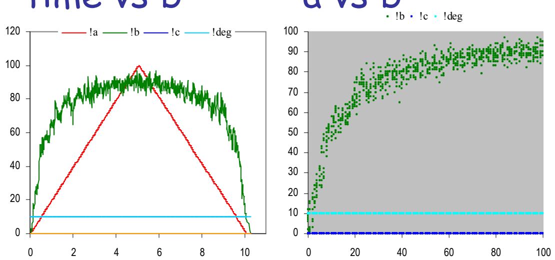

```
directive sample 10.0 1000
directive plot !a; !b; !c; !r

new a@1.0:chan new r@1.0:chan new b@1.0:chan new
c@1.0:chan

(*
let Amp_hi(a:chan, b:chan) =
  do !b; Amp_hi(a,b)
  or delay@1.0; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; Amp_hi(a,b)
*)

(*
let Amp_lo(a:chan, r:chan, b:chan) =
  do !r; Amp_lo(a,r,b)
  or ?a; Amp_hi(a,r,b)
and Amp_hi(a:chan, r:chan, b:chan) =
  do !b; Amp_hi(a,r,b)
  or ?r; Amp_lo(a,r,b)
  or delay@1.0; Amp_lo(a,r,b)
*)

let Amp2_lo(a:chan, r:chan, b:chan) =
  do !r; Amp2_lo(a,r,b)
  or ?a; Amp2_mi(a,r,b)
and Amp2_mi(a:chan, r:chan, b:chan) =
  do ?a; Amp2_hi(a,r,b)
  or ?r; Amp2_lo(a,r,b)
  or delay@1.0; Amp2_lo(a,r,b)
and Amp2_hi(a:chan, r:chan, b:chan) =
  do !b; Amp2_hi(a,r,b)
  or ?r; Amp2_lo(a,r,b)
  or delay@1.0; Amp2_mi(a,r,b)

run 100 of Amp2_lo(a,r,b)
(* run 100 of (Amp_lo(a,b) | Amp_lo(b,c)) *)

let clock(t:float, tick:chan) =       (* sends a tick every t
time *)
  (val ti = t/100.0 val d = 1.0/ti     (* by 100-step erlang
timers *)
   let step(n:int) = if n<=0 then !tick; clock(t,tick) else
delay@d; step(n-1)
   run step(100))
let S(a:chan, tick:chan) =
  do !a; S(a,tick) or ?tick; (S(a,tick) | S(a,tick))
let raising(a:chan, t:float) =
  (new tick:chan run (clock(t,tick) | S(a,tick)))
let S1(a:chan, tock:chan) =  do !a; S1(a,tock) or ?tock; ()
let SN(n:int, t:float, a:chan, tick:chan, tock:chan) =
  if n=0 then clock(t, tock) else ?tick; (S1(a,tock) | SN(n-
1,t,a,tick,tock))
let raisingfalling(a:chan, n:int, t:float) =
  (new tick:chan new tock:chan
   run (clock(t,tick) | SN(n,t,a,tick,tock)))

run raisingfalling(a,100,0.05)
```

Excitation Cascade with Degradation



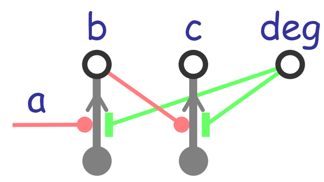
time vs b

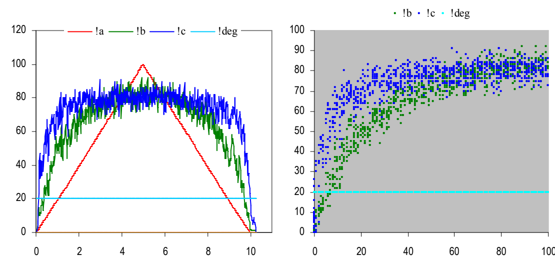
a vs b



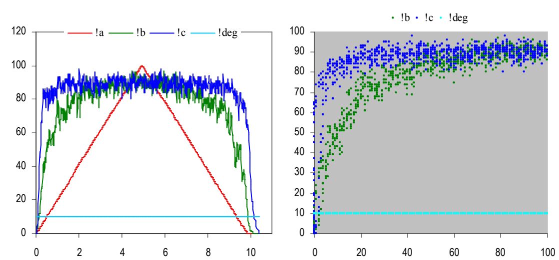
100*b vs 10*deg

```
directive sample 100.0 1000
directive plot !a; !b; !deg

new a@1.0:chan new b@1.0:chan new deg@1.0:chan
new _up:chan new _dn:chan
val del = 1.0

let Mon_hi(hi:chan, up:chan, dn:chan) =
  do !hi; Mon_hi(hi,up,dn) or ?dn; Mon_lo(hi,up,dn)
and Mon_lo(hi:chan, up:chan, dn:chan) =
  ?up; Mon_hi(hi,up,dn)

run (100 of Mon_lo(b,a,deg) | 10 of Mon_hi(deg,_up,_dn))

new tick:chan
let clock(t:float) =       (* sends a tick every t time *)
  (val ti = t/100.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t) else delay@d; step(n-1)
   run step(100))

let S() = do !a; S() or ?tick; (S() | S())

run 1 of (clock(3.0) | S())
```

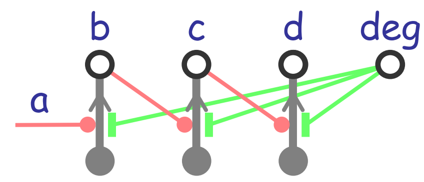


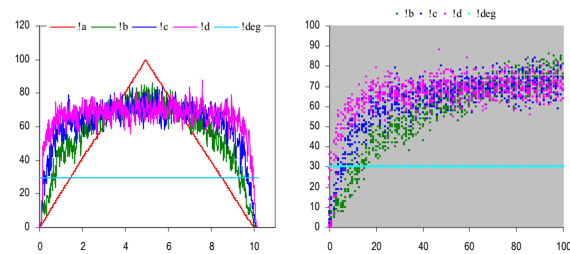


100*b,c vs 20*deg

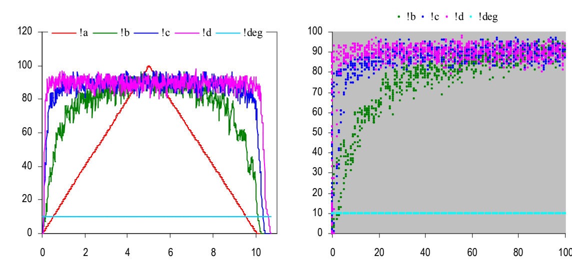


100*b,c vs 10*deg





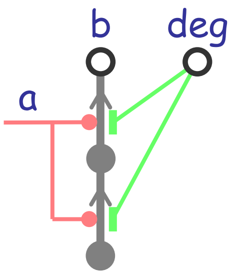
100*a,b,c vs 30*deg



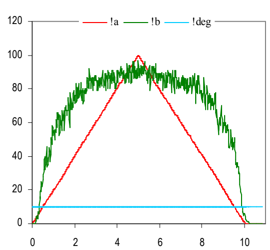
100*a,b,c vs 10^deg

no “sigma” response

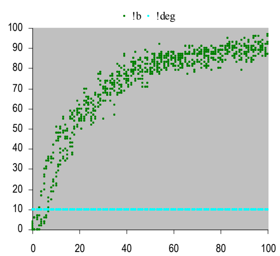
Double Excitation Cascade with Degradation



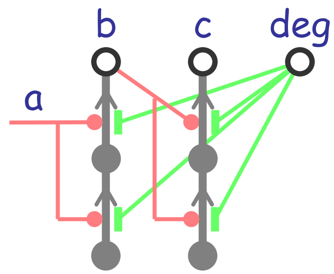
100*b vs 10*deg



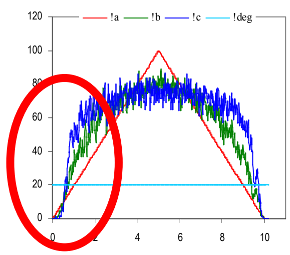
time vs b

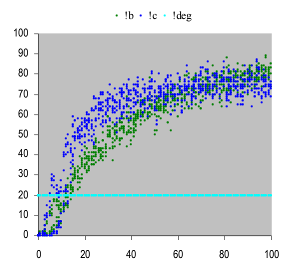


a vs b



100*b,c vs 20*deg





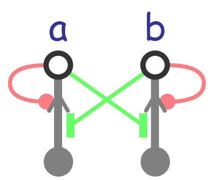
"sigma" response due to Erlang process

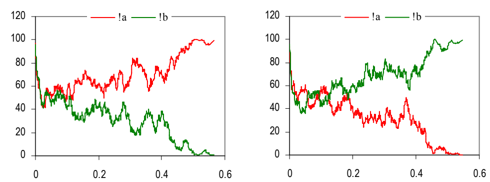
```
directive sample 100.0 1000
directive plot !a; !b; !c; !deg

new a@1.0:chan new b@1.0:chan new c@1.0:chan new deg@1.0:chan
new _up:chan new _dn:chan

let Mon_hi(hi:chan, up:chan, dn:chan) =
  do !hi; Mon_hi(hi,up,dn) or ?dn; Mon_mi(hi,up,dn)
and Mon_mi(hi:chan, up:chan, dn:chan) =
  do ?up; Mon_hi(hi,up,dn) or ?dn; Mon_lo(hi,up,dn)
and Mon_lo(hi:chan, up:chan, dn:chan) =
  ?up; Mon_mi(hi,up,dn)

(* run (100 of Mon_lo(b,a,deg) | 10 of Mon_hi(deg,_up,_dn)) *)
run (100 of (Mon_lo(b,a,deg) | Mon_lo(c,b,deg)) | 20 of Mon_hi(deg,_up,_dn))

let clock(tick:chan, t:float) =       (* sends a tick every t time *)
  (val ti = t/100.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(tick,t) else delay@d; step(n-1)
   run step(100))

new tick:chan
let S() = do !a; S() or ?tick; (S() | S())
run 1 of (clock(tick,3.0) | S())
```

Multistables and Oscillators

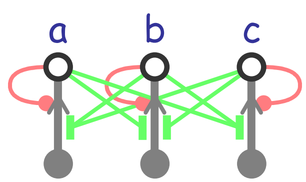
Monopolin Multistables

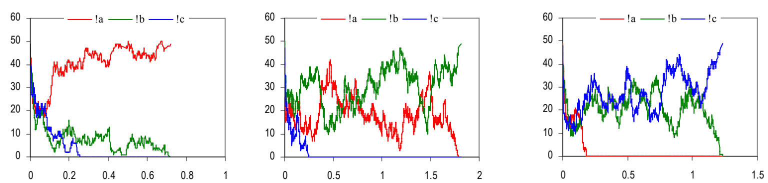
Each stimulates self and inhibits others





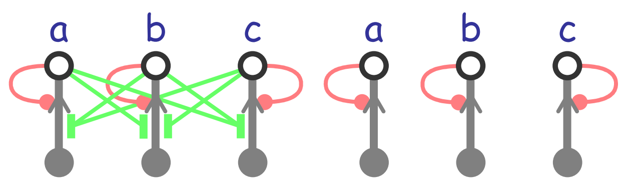

```
directive sample 5.0 10000
directive plot !a; !b

new a@1.0:chan new b@1.0:chan

let A_hi() = do !a; A_hi() or ?b; A_lo()
and A_lo() = ?a; A_hi()

let B_hi() = do !b; B_hi() or ?a; B_lo()
and B_lo() = ?b; B_hi()

run 100 of (A_hi() | B_hi())
```

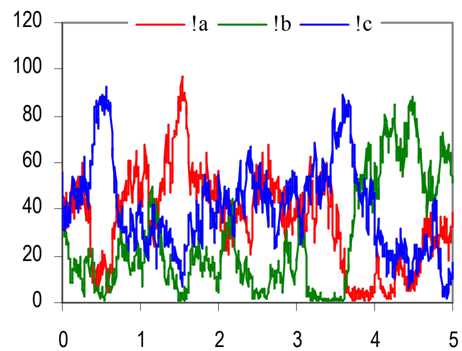




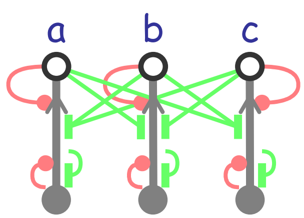

```
directive sample 5.0 10000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let A_hi() = do !a; A_hi() or ?b; A_lo() or ?c; A_lo()
and A_lo() = ?a; A_hi()

let B_hi() = do !b; B_hi() or ?c; B_lo() or ?a; B_lo()
and B_lo() = ?b; B_hi()

let C_hi() = do !c; C_hi() or ?a; C_lo() or ?b; C_lo()
and C_lo() = ?c; C_hi()

run 50 of (A_hi() | B_hi() | C_hi())
```

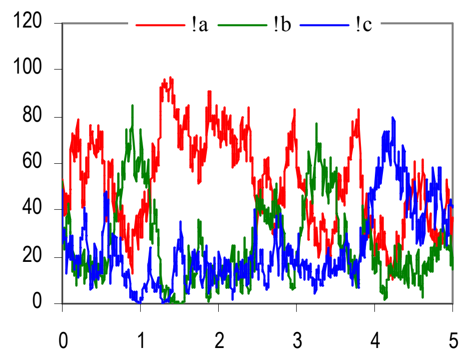
Mulstistables with Noise






```
directive sample 5.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let A_hi() = do !a; A_hi() or ?b; A_lo() or ?c; A_lo()
and A_lo() = ?a; A_hi()

let B_hi() = do !b; B_hi() or ?c; B_lo() or ?a; B_lo()
and B_lo() = ?b; B_hi()

let C_hi() = do !c; C_hi() or ?a; C_lo() or ?b; C_lo()
and C_lo() = ?c; C_hi()

let An() = !a; An()
and Bn() = !b; Bn()
and Cn() = !c; Cn()

run 100 of (A_hi() | B_hi() | C_hi())
run (An() | Bn() | Cn())
```

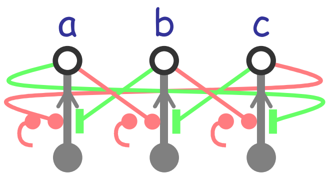




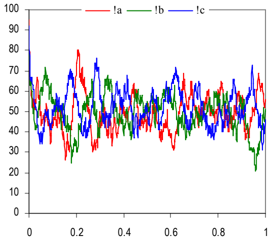

```
directive sample 5.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val noise = 1.0

let A_hi() = do !a; A_hi() or ?b; A_lo() or ?c; A_lo() or delay@noise; A_lo()
and A_lo() = do ?a; A_hi() or delay@noise; A_hi()

let B_hi() = do !b; B_hi() or ?c; B_lo() or ?a; B_lo() or delay@noise; B_lo()
and B_lo() = do ?b; B_hi() or delay@noise; B_hi()

let C_hi() = do !c; C_hi() or ?a; C_lo() or ?b; C_lo() or delay@noise; C_lo()
and C_lo() = do ?c; C_hi() or delay@noise; C_hi()

run 100 of (A_hi() | B_hi() | C_hi())
```

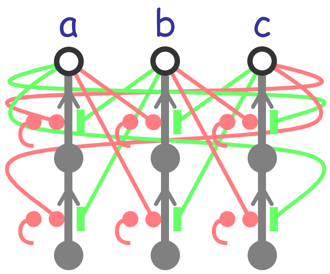
Monopolin Oscillators

Each stimulates the next and inhibits the previous.

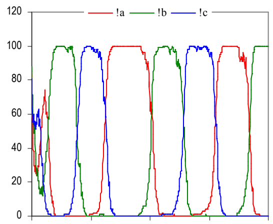


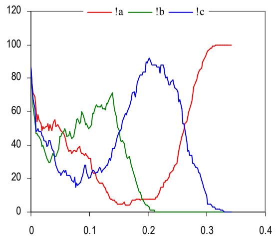

```
directive sample 1.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let A_hi() = do !a; A_hi() or ?b; A_lo()
and A_lo() = do ?a; A_hi() or delay@1.0; A_hi()

let B_hi() = do !b; B_hi() or ?c; B_lo()
and B_lo() = do ?b; B_hi() or delay@1.0; B_hi()

let C_hi() = do !c; C_hi() or ?a; C_lo()
and C_lo() = do ?c; C_hi() or delay@1.0; C_hi()

run 100 of (A_hi() | B_hi() | C_hi())
```






```
directive sample 1.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let A_hi() = do !a; A_hi() or ?b; A_mi()
and A_mi() = do ?a; A_hi() or ?b; A_lo() or
delay@1.0; A_hi()
and A_lo() = do ?a; A_mi() or delay@1.0; A_mi()

let B_hi() = do !b; B_hi() or ?c; B_mi()
and B_mi() = do ?b; B_hi() or ?c; B_lo() or
delay@1.0; B_hi()
and B_lo() = do ?b; B_mi() or delay@1.0; B_mi()

let C_hi() = do !c; C_hi() or ?a; C_mi()
and C_mi() = do ?c; C_hi() or ?a; C_lo() or
delay@1.0; C_hi()
and C_lo() = do ?c; C_mi() or delay@1.0; C_mi()

run 100 of (A_hi() | B_hi() | C_hi())
```



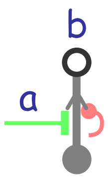


Without up-accretion:
deadlock

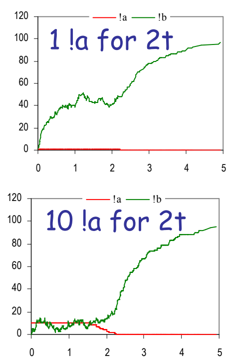
Inverters

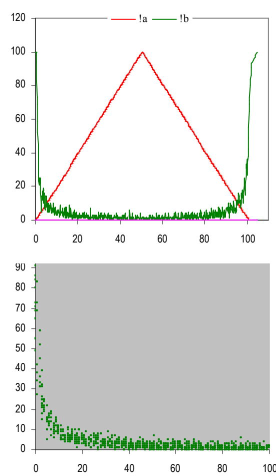
Pushup Inverter

Good logic needs a good inverter



Inv

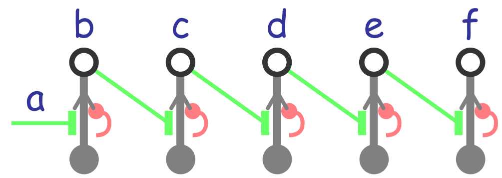


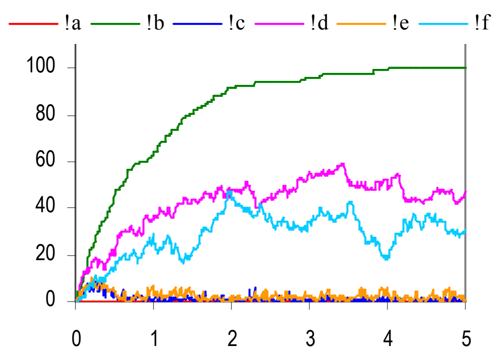


no hysteresis

```
directive sample 5.0 1000
directive plot !a; !b

new a@1.0:chan new b@1.0:chan
val del = 1.0

let Not_hi(a:chan, b:chan) =
  do !b; Not_hi(a,b) or ?a; Not_lo(a,b)
and Not_lo(a:chan, b:chan) =
  delay@del;Not_hi(a,b)

run 100 of Not_lo(a,b)

new tick:chan
let clock(t:float) =      (* sends a tick every t time *)
  (val ti = t/100.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t) else delay@d; step(n-1)
   run step(100))

let S() = do !a; S() or ?tick; ()

run 10 of (clock(2.0) | S())
```

…that alternates in cascades

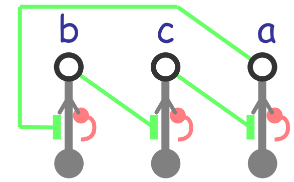


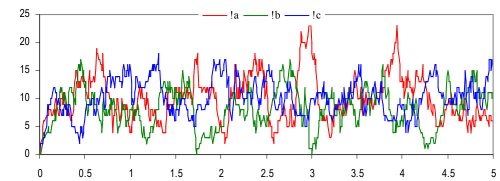


poor alternation

```
directive sample 5.0 1000
directive plot !a; !b; !c; !d; !e; !f

new a@1.0:chan new b@1.0:chan
new c@1.0:chan new d@1.0:chan
new e@1.0:chan new f@1.0:chan
val del = 1.0

let Not_hi(a:chan, b:chan) =
  do !b; Not_hi(a,b) or ?a; Not_lo(a,b)
and Not_lo(a:chan, b:chan) =
  delay@del;Not_hi(a,b)

run 100 of (Not_lo(a,b) | Not_lo(b,c)
| Not_lo(c,d) | Not_lo(d,e) | Not_lo(e,f))
```

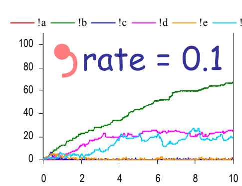
…that oscillates in odd cycles

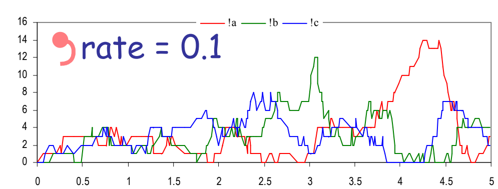




no oscillation

fiddling with rates does not seem to change the picture



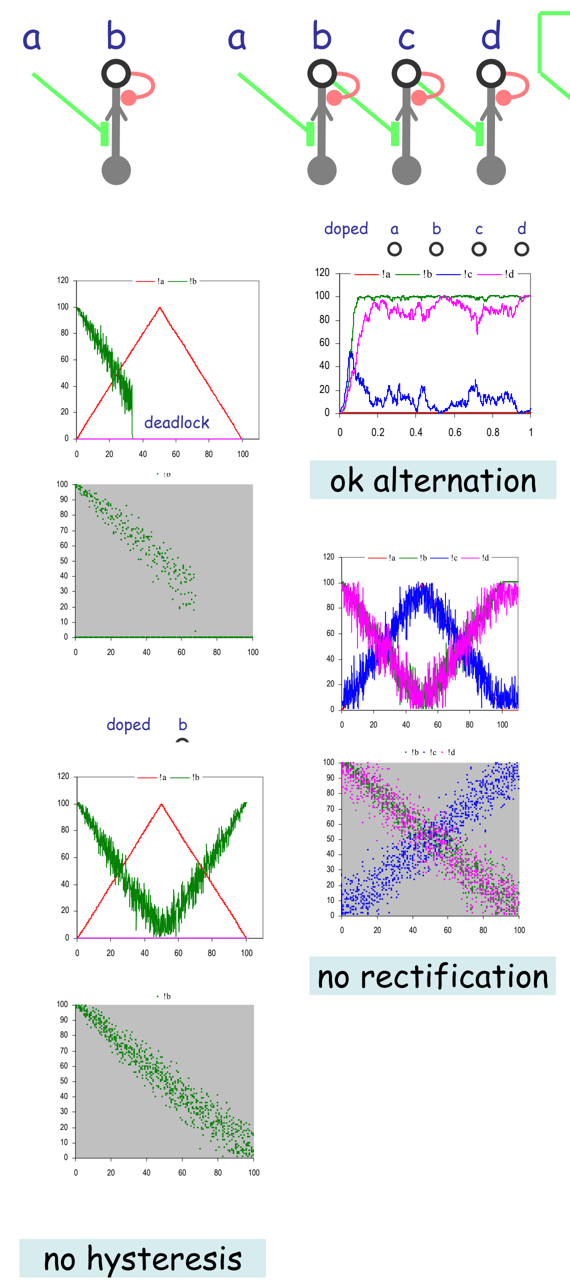


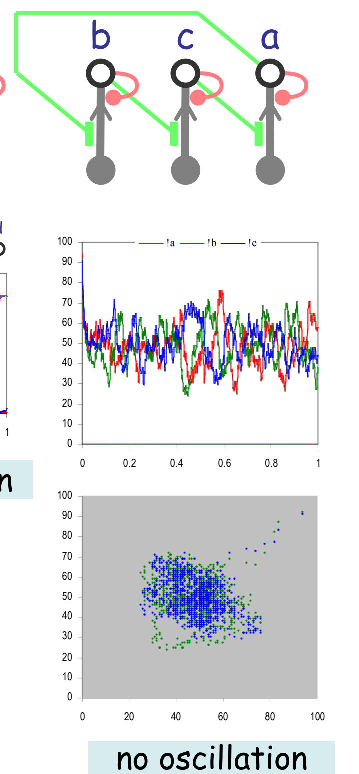

```
directive sample 5.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let Not_hi(a:chan, b:chan) =
  do !b; Not_hi(a,b) or ?a; Not_lo(a,b)
and Not_lo(a:chan, b:chan) =
  delay@del;Not_hi(a,b)

run 100 of (Not_lo(a,b) | Not_lo(b,c) | Not_lo(c,a))
```

Pullup Inverter (deadlocking low)

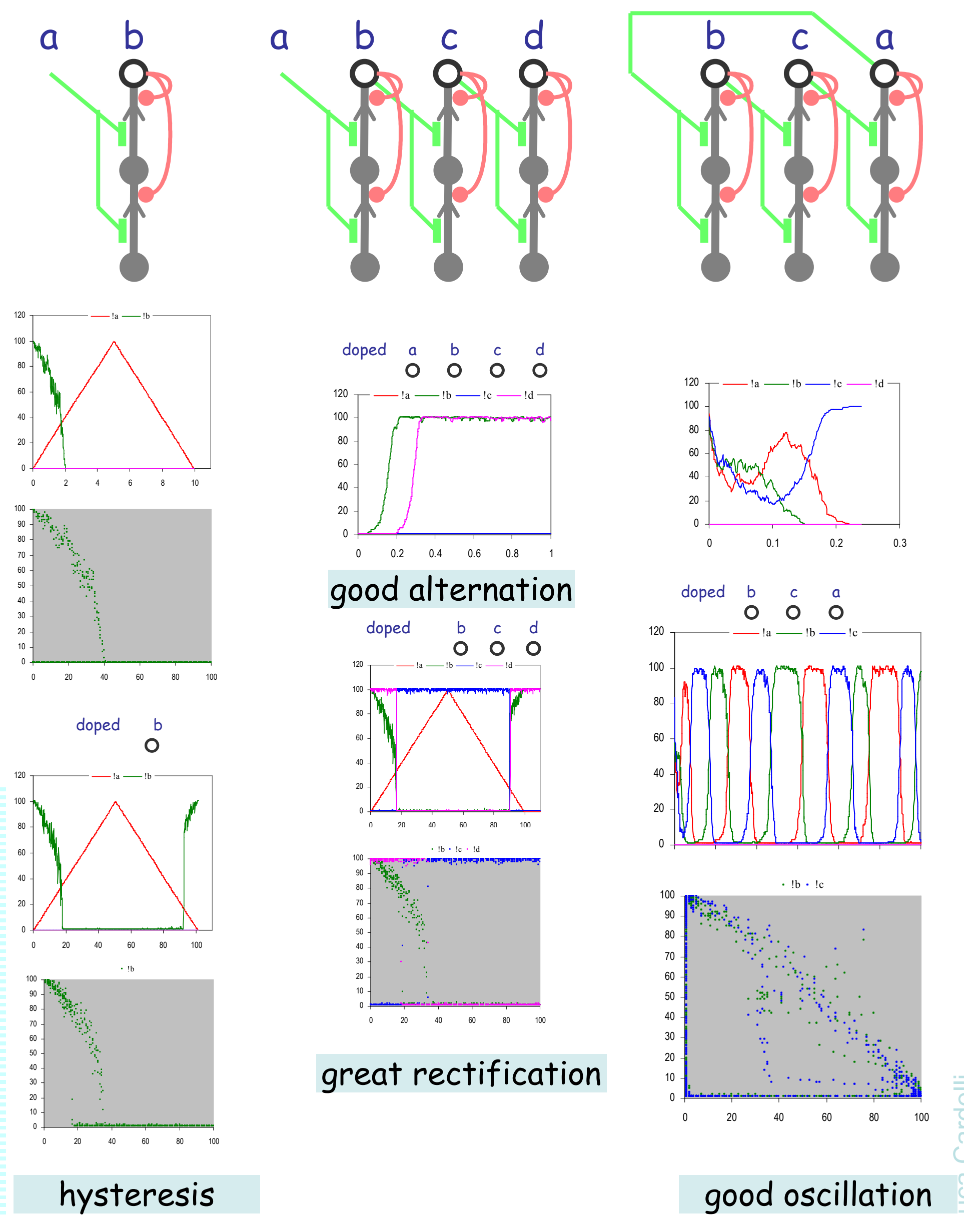
Aux

Aux

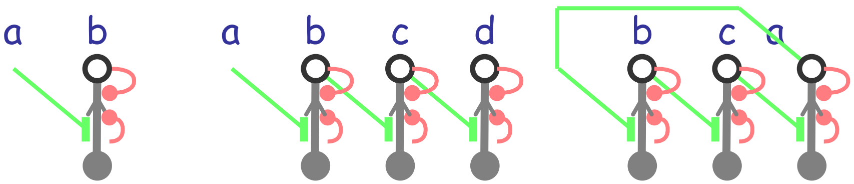


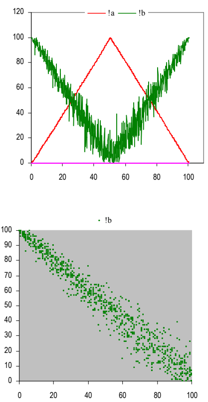



```
directive sample 100.0 1000
directive plot !a; !b; !c; !d

new a@1.0:chan new b@1.0:chan new c@1.0:chan new d@1.0:chan

let Inv_lo(a:chan, b:chan) =
  ?b; Inv_hi(a,b)
and Inv_hi(a:chan, b:chan) =
  do !b; Inv_hi(a,b)
  or ?a; Inv_lo(a,b)

let Inv2_lo(a:chan, b:chan) =
  ?b; Inv2_mi(a,b)
and Inv2_mi(a:chan, b:chan) =
  do ?b; Inv2_hi(a,b)
  or ?a; Inv2_lo(a,b)
and Inv2_hi(a:chan, b:chan) =
  do !b; Inv2_hi(a,b)
  or ?a; Inv2_mi(a,b)

run 100 of (Inv2_hi(a,b) | Inv2_lo(b,c))
(* run 100 of Inv2_hi(a,b) *)

let clock(t:float, tick:chan) =        (* sends a tick every t time *)
  (val ti = t/100.0 val d = 1.0/ti     (* by 100-step erlang timers *)
   let step(n:int) = if n<=0 then !tick; clock(t,tick) else delay@d; step(n-1)
   run step(100))
let S1(a:chan, tock:chan) =  do !a; S1(a,tock) or ?tock; ()
let SN(n:int, t:float, a:chan, tick:chan, tock:chan) =
  if n=0 then clock(t, tock) else ?tick; (S1(a,tock) | SN(n-1,t,a,tick,tock))
let raisingfalling(a:chan, n:int, t:float) =
  (new tick:chan new tock:chan
   run (clock(t,tick) | SN(n,t,a,tick,tock)))

let K(a:chan) = !a; K(a)
(* run (K(a) | K(b) | K(c) | K(d)) *)
run (K(b) | K(c))

run raisingfalling(a,100,0.5)
```

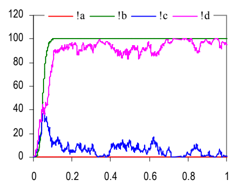


Pushup/Pullup Inverter

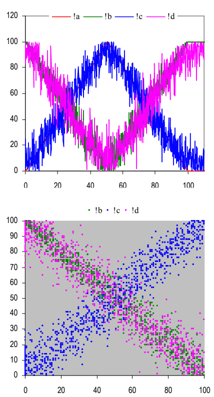




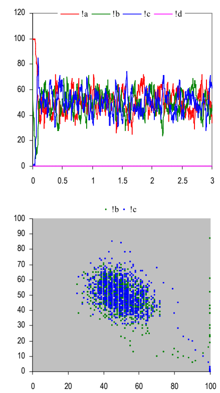
no hysteresis



poor alternation

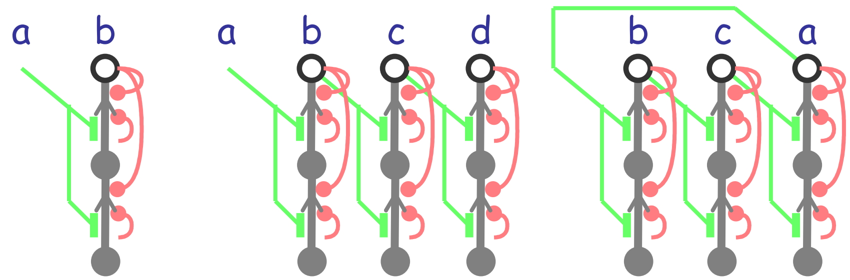


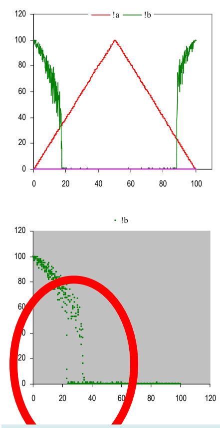
no rectification



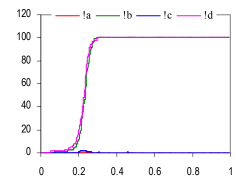
no oscillation

```
directive sample 100.0 1000
directive plot !a; !b; !c; !d

new a@1.0:chan new b@1.0:chan new c@1.0:chan new d@1.0:chan

let Inv_lo(a:chan, b:chan) =
  do ?b; Inv_hi(a,b)
  or delay@1.0; Inv_hi(a,b)
and Inv_hi(a:chan, b:chan) =
  do !b; Inv_hi(a,b)
  or ?a; Inv_lo(a,b)

let Inv2_lo(a:chan, b:chan) =
  do ?b; Inv2_mi(a,b)
  or delay@1.0; Inv2_mi(a,b)
and Inv2_mi(a:chan, b:chan) =
  do ?b; Inv2_hi(a,b)
  or delay@1.0; Inv2_hi(a,b)
  or ?a; Inv2_lo(a,b)
and Inv2_hi(a:chan, b:chan) =
  do !b; Inv2_hi(a,b)
  or ?a; Inv2_mi(a,b)

run 100 of (Inv2_hi(a,b) | Inv2_lo(b,c))
(* run 100 of Inv2_hi(a,b) *)

let clock(t:float, tick:chan) =        (* sends a tick every t time *)
  (val ti = t/100.0 val d = 1.0/ti     (* by 100-step erlang timers *)
   let step(n:int) = if n<=0 then !tick; clock(t,tick) else delay@d; step(n-1)
   run step(100))
let S1(a:chan, tock:chan) =  do !a; S1(a,tock) or ?tock; ()
let SN(n:int, t:float, a:chan, tick:chan, tock:chan) =
  if n=0 then clock(t, tock) else ?tick; (S1(a,tock) | SN(n-1,t,a,tick,tock))
let raisingfalling(a:chan, n:int, t:float) =
  (new tick:chan new tock:chan
   run (clock(t,tick) | SN(n,t,a,tick,tock)))

let K(a:chan) = !a; K(a)
(* run (K(a) | K(b) | K(c) | K(d)) *)

run raisingfalling(a,100,0.5)
```

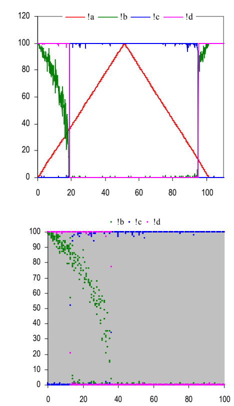




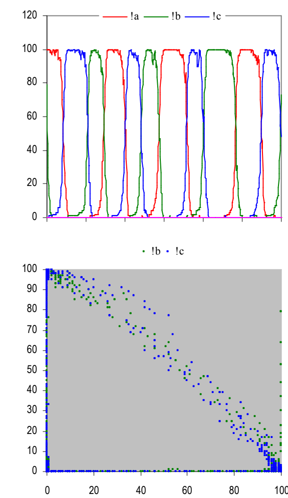
hysteresis



great alternation



great rectification

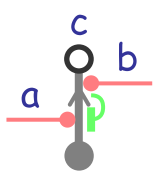


great oscillation

Boolean Gates

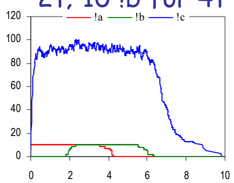
Monopolin Boolean Gates

A "monopolin signal" consists of either the *presence* of a certain pole (designated "hi") in state *current*, or the *absence* of such a pole.

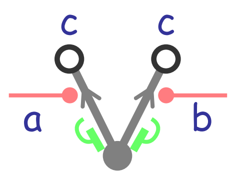


Or

10 !a for 4t
2t; 10 !b for 4t

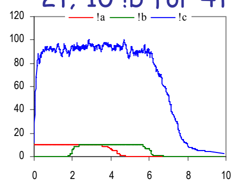



```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let Or_hi(a:chan, b:chan, c:chan) =
  do !c; Or_hi(a,b,c) or delay@del; Or_lo(a,b,c)
and Or_lo(a:chan, b:chan, c:chan) =
  do ?a; Or_hi(a,b,c) or ?b; Or_hi(a,b,c)

run 100 of Or_lo(a,b,c)

let clock(t:float, tick:chan) =       (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

run 10 of (new tick:chan run (clock(4.0,tick) | S_a(tick)))
run 10 of (new tick:chan run (clock(2.0,tick) | S_b(tick)))
```

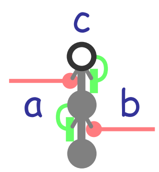


split-Or

10 !a for 4t
2t; 10 !b for 4t

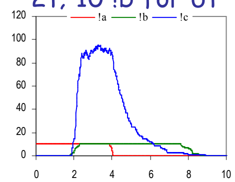



```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let Or_hi_a(a:chan, b:chan, c:chan) =
  do !c; Or_hi_a(a,b,c) or delay@del; Or_lo(a,b,c)
and Or_hi_b(a:chan, b:chan, c:chan) =
  do !c; Or_hi_b(a,b,c) or delay@del; Or_lo(a,b,c)
and Or_lo(a:chan, b:chan, c:chan) =
  do ?a; Or_hi_a(a,b,c) or ?b; Or_hi_b(a,b,c)

run 100 of Or_lo(a,b,c)

let clock(t:float, tick:chan) =       (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

run 10 of (new tick:chan run (clock(4.0,tick) | S_a(tick)))
run 10 of (new tick:chan run (clock(2.0,tick) | S_b(tick)))
```

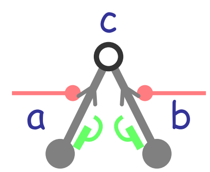


And
0001

10 !a for 4t
2t; 10 !b for 6t

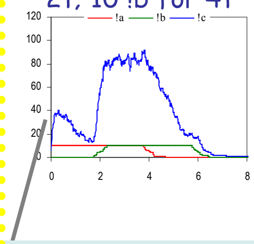



```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let And_hi(a:chan, b:chan, c:chan) =
  do !c; And_hi(a,b,c) or delay@del; And_lo_a(a,b,c)
and And_lo_a(a:chan, b:chan, c:chan) =
  do ?a; And_hi(a,b,c) or delay@del; And_lo_b(a,b,c)
and And_lo_b(a:chan, b:chan, c:chan) =
  ?b; And_lo_a(a,b,c)

run 100 of And_lo_b(a,b,c)

let clock(t:float, tick:chan) =       (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; S_b3(tick)
and S_b3(tick:chan) = do !b; S_b3(tick) or ?tick; ()

run 10 of (new tick:chan run (clock(4.0,tick) | S_a(tick)))
run 10 of (new tick:chan run (clock(2.0,tick) | S_b(tick)))
```



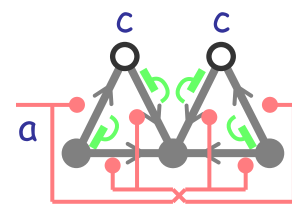
split-And

10 !a for 4t
2t; 10 !b for 4t



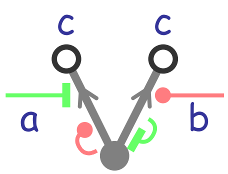
glitch on a-up

```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let And_hi(a:chan, b:chan, c:chan) =
  do !c; And_hi(a,b,c) or delay@del; And_lo_a(a,b,c)
  or delay@del; And_lo_b(a,b,c)
and And_lo_a(a:chan, b:chan, c:chan) =
  ?a; And_hi(a,b,c)
and And_lo_b(a:chan, b:chan, c:chan) =
  ?b; And_hi(a,b,c)

run 50 of (And_lo_a(a,b,c) | And_lo_b(a,b,c))

let clock(t:float, tick:chan) =       (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

run 10 of (new tick:chan run (clock(4.0,tick) | S_a(tick)))
run 10 of (new tick:chan run (clock(2.0,tick) | S_b(tick)))
```

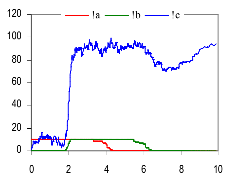


Xor

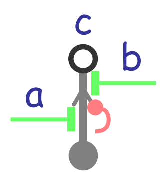
Monopolin Boolean Gates



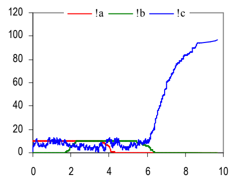
Imply



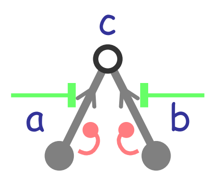

```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let Imply_hi_a(a:chan, b:chan, c:chan) =
  do !c; Imply_hi_a(a,b,c) or ?a; Imply_lo(a,b,c)
and Imply_hi_b(a:chan, b:chan, c:chan) =
  do !c; Imply_hi_b(a,b,c) or delay@del; Imply_lo(a,b,c)
and Imply_lo(a:chan, b:chan, c:chan) =
  do ?b; Imply_hi_b(a,b,c) or delay@del; Imply_hi_a(a,b,c)

run 100 of Imply_lo(a,b,c)

let clock(t:float, tick:chan) =      (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

run 10 of (new tick:chan run (clock(4.0,tick) | S_a(tick)))
run 10 of (new tick:chan run (clock(2.0,tick) | S_b(tick)))
```



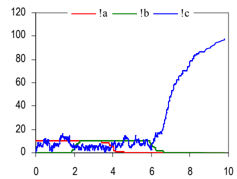
Nor
1000



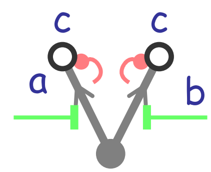

```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let Nor_hi(a:chan, b:chan, c:chan) =
  do !c; Nor_hi(a,b,c) or ?a; Nor_lo(a,b,c) or ?b; Nor_lo(a,b,c)
and Nor_lo(a:chan, b:chan, c:chan) =
  delay@del; Nor_hi(a,b,c)

run 100 of Nor_lo(a,b,c)

let clock(t:float, tick:chan) =      (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

run 10 of (new tick:chan run (clock(4.0,tick) | S_a(tick)))
run 10 of (new tick:chan run (clock(2.0,tick) | S_b(tick)))
```



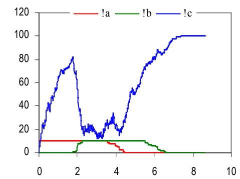
split-Nor



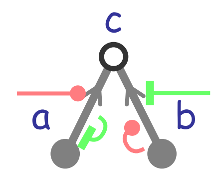

```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let Nor_hi(a:chan, b:chan, c:chan) =
  do !c; Nor_hi(a,b,c) or ?a; Nor_lo_a(a,b,c) or ?b;
Nor_lo_b(a,b,c)
and Nor_lo_a(a:chan, b:chan, c:chan) =
  delay@del; Nor_hi(a,b,c)
and Nor_lo_b(a:chan, b:chan, c:chan) =
  delay@del; Nor_hi(a,b,c)

run 50 of (Nor_lo_a(a,b,c) | Nor_lo_b(a,b,c))

let clock(t:float, tick:chan) =      (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

run 10 of (new tick:chan run (clock(4.0,tick) | S_a(tick)))
run 10 of (new tick:chan run (clock(2.0,tick) | S_b(tick)))
```



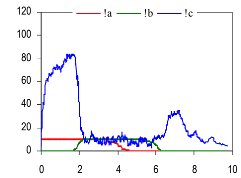
Nand



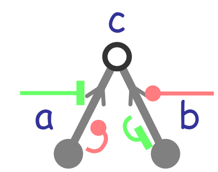

```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let Nand_hi_a(a:chan, b:chan, c:chan) =
  do !c; Nand_hi_a(a,b,c) or ?a; Nand_lo(a,b,c)
and Nand_hi_b(a:chan, b:chan, c:chan) =
  do !c; Nand_hi_b(a,b,c) or ?b; Nand_lo(a,b,c)
and Nand_lo(a:chan, b:chan, c:chan) =
  do delay@del; Nand_hi_a(a,b,c) or delay@del; Nand_hi_b(a,b,c)

run 100 of Nand_lo(a,b,c)

let clock(t:float, tick:chan) =      (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

run 10 of (new tick:chan run (clock(4.0,tick) | S_a(tick)))
run 10 of (new tick:chan run (clock(2.0,tick) | S_b(tick)))
```



0010



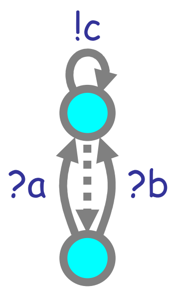

```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let OOlO_hi(a:chan, b:chan, c:chan) =
  do !c; OOlO_hi(a,b,c) or delay@del; OOlO_lo_a(a,b,c) or ?b;
OOlO_lo_b(a,b,c)
and OOlO_lo_a(a:chan, b:chan, c:chan) =
  ?a; OOlO_hi(a,b,c)
and OOlO_lo_b(a:chan, b:chan, c:chan) =
  delay@del; OOlO_hi(a,b,c)

run 50 of (OOlO_lo_a(a,b,c) | OOlO_lo_b(a,b,c))

let clock(t:float, tick:chan) =      (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

run 10 of (new tick:chan run (clock(4.0,tick) | S_a(tick)))
run 10 of (new tick:chan run (clock(2.0,tick) | S_b(tick)))
```



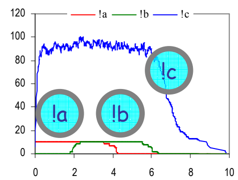
0100

Boolean Gates: The Automata View

c = a or b

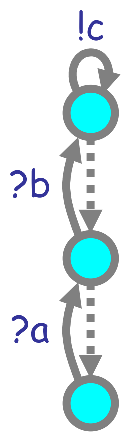


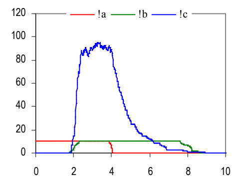
Inputs:
10 !a for 4t
2t; 10 !b for 4t




```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let Or_hi(a:chan, b:chan, c:chan) =
  do !c; Or_hi(a,b,c) or delay@del; Or_lo(a,b,c)
and Or_lo(a:chan, b:chan, c:chan) =
  do ?a; Or_hi(a,b,c) or ?b; Or_hi(a,b,c)

run 100 of Or_lo(a,b,c)

let clock(t:float, tick:chan) =      (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

run 10 of (new tick:chan run (clock(4.0,tick) | S_a(tick)))
run 10 of (new tick:chan run (clock(2.0,tick) | S_b(tick)))
```

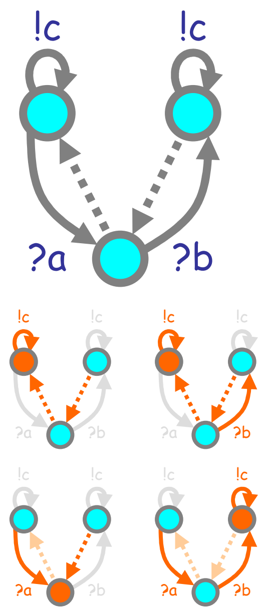
c = a and b

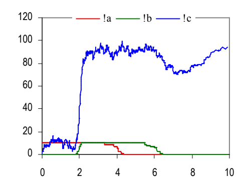





```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let And_hi(a:chan, b:chan, c:chan) =
  do !c; And_hi(a,b,c) or delay@del; And_lo_a(a,b,c)
and And_lo_a(a:chan, b:chan, c:chan) =
  do ?a; And_hi(a,b,c) or delay@del; And_lo_b(a,b,c)
and And_lo_b(a:chan, b:chan, c:chan) =
  ?b; And_lo_a(a,b,c)

run 100 of And_lo_b(a,b,c)

let clock(t:float, tick:chan) =      (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; S_b3(tick)
and S_b3(tick:chan) = do !b; S_b3(tick) or ?tick; ()

run 10 of (new tick:chan run (clock(4.0,tick) | S_a(tick)))
run 10 of (new tick:chan run (clock(2.0,tick) | S_b(tick)))
```

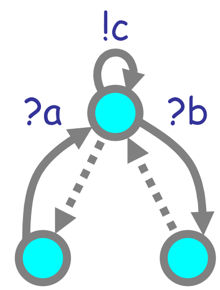
c = a imply b

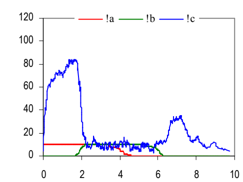





```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let Imply_hi_a(a:chan, b:chan, c:chan) =
  do !c; Imply_hi_a(a,b,c) or ?a; Imply_lo(a,b,c)
and Imply_hi_b(a:chan, b:chan, c:chan) =
  do !c; Imply_hi_b(a,b,c) or delay@del; Imply_lo(a,b,c)
and Imply_lo(a:chan, b:chan, c:chan) =
  do ?b; Imply_hi_b(a,b,c) or delay@del; Imply_hi_a(a,b,c)

run 100 of Imply_lo(a,b,c)

let clock(t:float, tick:chan) =      (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

run 10 of (new tick:chan run (clock(4.0,tick) | S_a(tick)))
run 10 of (new tick:chan run (clock(2.0,tick) | S_b(tick)))
```

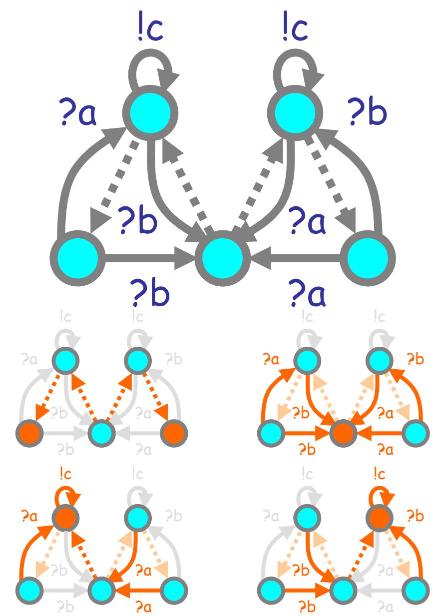
c = a unless b

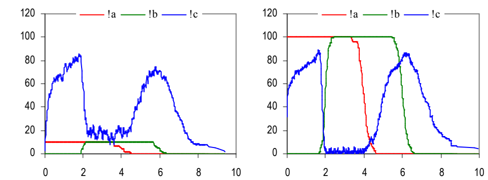





```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let OOIO_hi(a:chan, b:chan, c:chan) =
  do !c; OOIO_hi(a,b,c) or delay@del; OOIO_lo_a(a,b,c) or ?b;
OOIO_lo_b(a,b,c)
and OOIO_lo_a(a:chan, b:chan, c:chan) =
  ?a; OOIO_hi(a,b,c)
and OOIO_lo_b(a:chan, b:chan, c:chan) =
  delay@del; OOIO_hi(a,b,c)

run 50 of (OOIO_lo_a(a,b,c) | OOIO_lo_b(a,b,c))

let clock(t:float, tick:chan) =      (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

run 10 of (new tick:chan run (clock(4.0,tick) | S_a(tick)))
run 10 of (new tick:chan run (clock(2.0,tick) | S_b(tick)))
```

c = a xor b

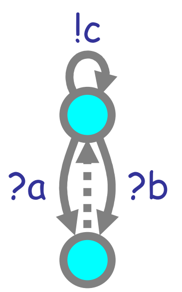


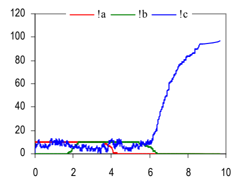



```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Xor_hi_a(a:chan, b:chan, c:chan) =
  do !c; Xor_hi_a(a,b,c) or ?b; Xor_lo_ab(a,b,c) or delay@1.0; Xor_lo_a(a,b,c)
and Xor_hi_b(a:chan, b:chan, c:chan) =
  do !c; Xor_hi_b(a,b,c) or ?a; Xor_lo_ab(a,b,c) or delay@1.0; Xor_lo_b(a,b,c)
and Xor_lo_a(a:chan, b:chan, c:chan) =
  do ?a; Xor_hi_a(a,b,c) or ?b; Xor_lo_ab(a,b,c)
and Xor_lo_b(a:chan, b:chan, c:chan) =
  do ?b; Xor_hi_b(a,b,c) or ?a; Xor_lo_ab(a,b,c)
and Xor_lo_ab(a:chan, b:chan, c:chan) =
  do delay@1.0; Xor_hi_a(a,b,c) or delay@1.0; Xor_hi_b(a,b,c)

run 50 of (Xor_lo_a(a,b,c) | Xor_lo_b(a,b,c))

let clock(t:float, tick:chan) =      (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

run 10 of (new tick:chan run (clock(4.0,tick) | S_a(tick)))
run 10 of (new tick:chan run (clock(2.0,tick) | S_b(tick)))
```

Boolean Gates: The Automata View

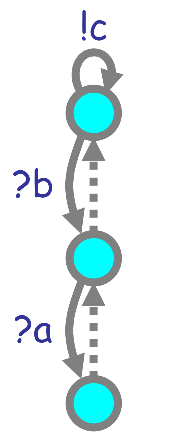
c = a nor b



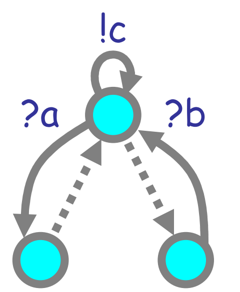



```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan
val del = 1.0

let Nor_hi(a:chan, b:chan, c:chan) =
  do !c; Nor_hi(a,b,c) or ?a; Nor_lo(a,b,c) or ?b; Nor_lo(a,b,c)
and Nor_lo(a:chan, b:chan, c:chan) =
  delay@del; Nor_hi(a,b,c)

run 100 of Nor_lo(a,b,c)

let clock(t:float, tick:chan) =      (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

run 10 of (new tick:chan run (clock(4.0,tick) | S_a(tick)))
run 10 of (new tick:chan run (clock(2.0,tick) | S_b(tick)))
```

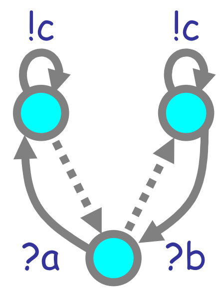
c = a nand b



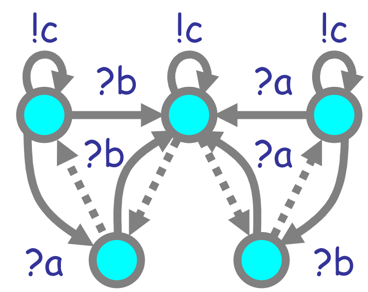
c = b unless a

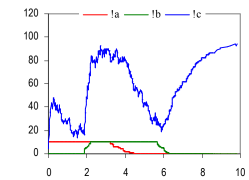


c = b imply a



c = a iff b



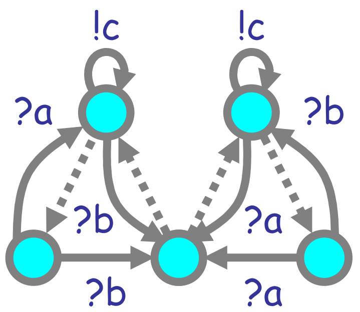



```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Iff_hi_a(a:chan, b:chan, c:chan) =
  do !c; Iff_hi_a(a,b,c) or ?a; Iff_lo_a(a,b,c) or ?b; Iff_hi_ab(a,b,c)
and Iff_hi_b(a:chan, b:chan, c:chan) =
  do !c; Iff_hi_b(a,b,c) or ?b; Iff_lo_b(a,b,c) or ?a; Iff_hi_ab(a,b,c)
and Iff_hi_ab(a:chan, b:chan, c:chan) =
  do !c; Iff_hi_ab(a,b,c) or delay@1.0; Iff_lo_a(a,b,c) or delay@1.0; Iff_lo_b(a,b,c)
and Iff_lo_a(a:chan, b:chan, c:chan) =
  do ?b; Iff_hi_ab(a,b,c) or delay@1.0; Iff_hi_a(a,b,c)
and Iff_lo_b(a:chan, b:chan, c:chan) =
  do ?a; Iff_hi_ab(a,b,c) or delay@1.0; Iff_hi_b(a,b,c)

run 50 of (Iff_lo_a(a,b,c) | Iff_lo_b(a,b,c))

let clock(t:float, tick:chan) =      (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

run 10 of (new tick:chan run (clock(4.0,tick) | S_a(tick)))
run 10 of (new tick:chan run (clock(2.0,tick) | S_b(tick)))
```

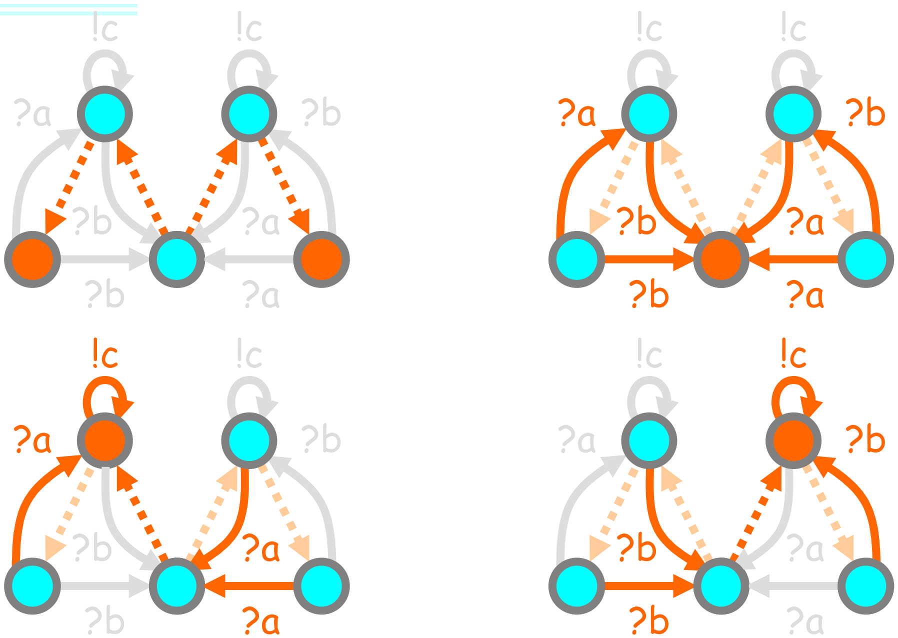
Xor and OpAmp

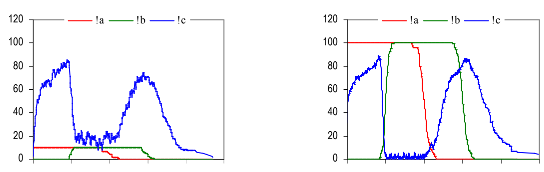
Xor in Detail

```
directive sample 10.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Xor_hi_a(a:chan, b:chan, c:chan) =
  do !c; Xor_hi_a(a,b,c) or ?b; Xor_lo_ab(a,b,c) or delay@1.0; Xor_lo_a(a,b,c)
and Xor_hi_b(a:chan, b:chan, c:chan) =
  do !c; Xor_hi_b(a,b,c) or ?a; Xor_lo_ab(a,b,c) or delay@1.0; Xor_lo_b(a,b,c)
and Xor_lo_a(a:chan, b:chan, c:chan) =
  do ?a; Xor_hi_a(a,b,c) or ?b; Xor_lo_ab(a,b,c)
and Xor_lo_b(a:chan, b:chan, c:chan) =
  do ?b; Xor_hi_b(a,b,c) or ?a; Xor_lo_ab(a,b,c)
and Xor_lo_ab(a:chan, b:chan, c:chan) =
  do delay@1.0; Xor_hi_a(a,b,c) or delay@1.0; Xor_hi_b(a,b,c)

run 50 of (Xor_lo_a(a,b,c) | Xor_lo_b(a,b,c))

let clock(t:float, tick:chan) =      (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

run 10 of (new tick:chan run (clock(4.0,tick) | S_a(tick)))
run 10 of (new tick:chan run (clock(2.0,tick) | S_b(tick)))
```

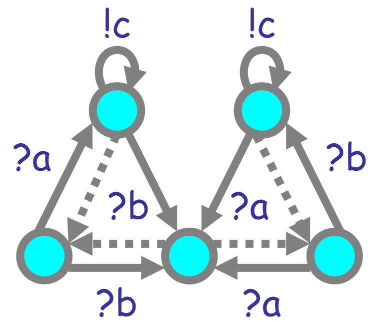


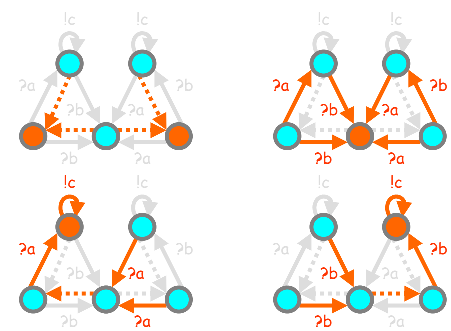
c = a xor b



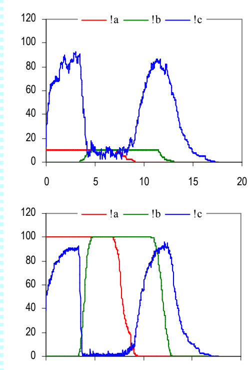


a different xor





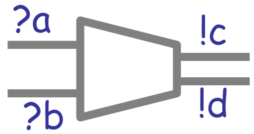
```
directive sample 20.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Xor_hi_a(a:chan, b:chan, c:chan) =
  do !c; Xor_hi_a(a,b,c) or ?b; Xor_lo_ab(a,b,c) or delay@1.0; Xor_lo_a(a,b,c)
and Xor_hi_b(a:chan, b:chan, c:chan) =
  do !c; Xor_hi_b(a,b,c) or ?a; Xor_lo_ab(a,b,c) or delay@1.0; Xor_lo_b(a,b,c)
and Xor_lo_a(a:chan, b:chan, c:chan) =
  do ?a; Xor_hi_a(a,b,c) or ?b; Xor_lo_ab(a,b,c)
and Xor_lo_b(a:chan, b:chan, c:chan) =
  do ?b; Xor_hi_b(a,b,c) or ?a; Xor_lo_ab(a,b,c)
and Xor_lo_ab(a:chan, b:chan, c:chan) =
  do delay@1.0; Xor_lo_a(a,b,c) or delay@1.0; Xor_lo_b(a,b,c)

run 50 of (Xor_lo_a(a,b,c) | Xor_lo_b(a,b,c))

let clock(t:float, tick:chan) =      (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

run 10 of (new tick:chan run (clock(8.0,tick) | S_a(tick)))
run 10 of (new tick:chan run (clock(4.0,tick) | S_b(tick)))
```

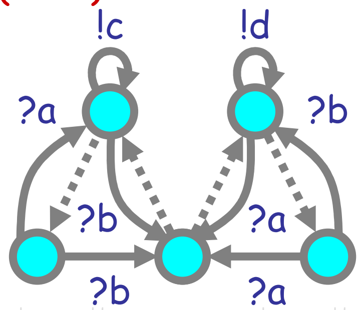


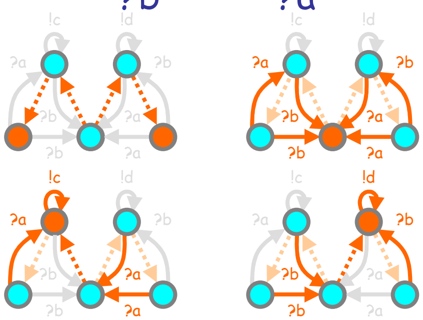
Luca Cardelli

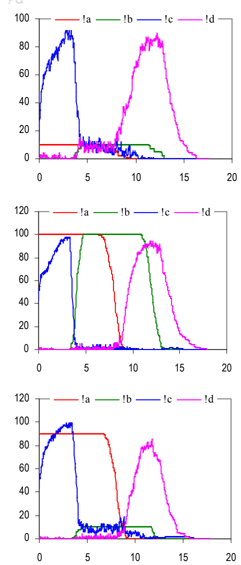
Xor as an Op Amp

c = A*(a - b)
d = A*(b - a)







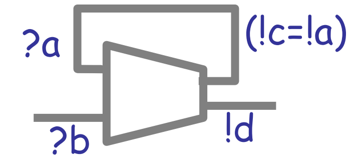



```
directive sample 20.0 1000
directive plot !a; !b; !c; !d

new a@1.0:chan new b@1.0:chan new c@1.0:chan new d@1.0:chan

let Xor_hi_a(a:chan, b:chan, c:chan, d:chan) =
  do !c; Xor_hi_a(a,b,c,d) or ?b; Xor_lo_ab(a,b,c,d) or delay@1.0; Xor_lo_a(a,b,c,d)
and Xor_hi_b(a:chan, b:chan, c:chan, d:chan) =
  do !d; Xor_hi_b(a,b,c,d) or ?a; Xor_lo_ab(a,b,c,d) or delay@1.0; Xor_lo_b(a,b,c,d)
and Xor_lo_a(a:chan, b:chan, c:chan, d:chan) =
  do ?a; Xor_hi_a(a,b,c,d) or ?b; Xor_lo_ab(a,b,c,d)
and Xor_lo_b(a:chan, b:chan, c:chan, d:chan) =
  do ?b; Xor_hi_b(a,b,c,d) or ?a; Xor_lo_ab(a,b,c,d)
and Xor_lo_ab(a:chan, b:chan, c:chan, d:chan) =
  do delay@1.0; Xor_hi_a(a,b,c,d) or delay@1.0; Xor_hi_b(a,b,c,d)

run 50 of (Xor_lo_a(a,b,c,d) | Xor_lo_b(a,b,c,d))

let clock(t:float, tick:chan) =      (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

run 100 of (new tick:chan run (clock(8.0,tick) | S_a(tick)))
run 100 of (new tick:chan run (clock(4.0,tick) | S_b(tick)))
```

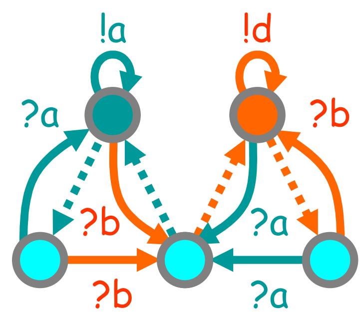
Follower (a standard OpAmp trick)

a=0 b=0 ⇒ d=b-a=0 a=c=a-b=0
a=0 b=1 ⇒ d=b-a=1 a=c=a-b=0
a=1 b=0 ⇒ d=b-a=0 a=c=a-b=1
a=1 b=1 ⇒ d=b-a=0 a=c=a-b=0
hence d=1 at next step

hence d=b

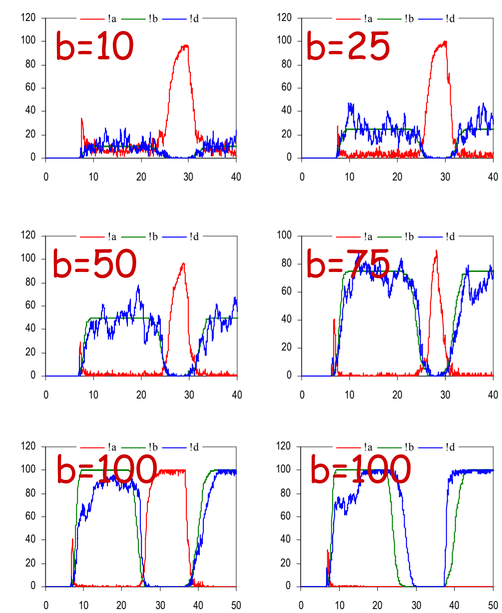


"Noninverting Configuration"




```
directive sample 40.0 1000
directive plot !a; !b; !d

new a@1.0:chan new b@1.0:chan new d@1.0:chan

let Xor_hi_a(a:chan, b:chan, c:chan, d:chan) =
  do !c; Xor_hi_a(a,b,c,d) or ?b; Xor_lo_ab(a,b,c,d) or delay@1.0; Xor_lo_a(a,b,c,d)
and Xor_hi_b(a:chan, b:chan, c:chan, d:chan) =
  do !d; Xor_hi_b(a,b,c,d) or ?a; Xor_lo_ab(a,b,c,d) or delay@1.0; Xor_lo_b(a,b,c,d)
and Xor_lo_a(a:chan, b:chan, c:chan, d:chan) =
  do ?a; Xor_hi_a(a,b,c,d) or ?b; Xor_lo_ab(a,b,c,d)
and Xor_lo_b(a:chan, b:chan, c:chan, d:chan) =
  do ?b; Xor_hi_b(a,b,c,d) or ?a; Xor_lo_ab(a,b,c,d)
and Xor_lo_ab(a:chan, b:chan, c:chan, d:chan) =
  do delay@1.0; Xor_hi_a(a,b,c,d) or delay@1.0; Xor_hi_b(a,b,c,d)

run 50 of (Xor_lo_a(a,b,a,d) | Xor_lo_b(a,b,a,d))

let clock(t:float, tick:chan) =      (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; S_b3(tick)
and S_b3(tick:chan) = ?tick; S_b4(tick)
and S_b4(tick:chan) = !b; S_b4(tick)

run 10 of (new tick:chan run (clock(8.0,tick) | S_b(tick)))
```

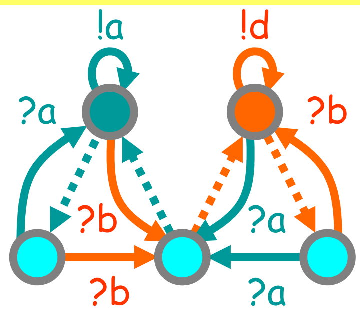
d=b analog response!!

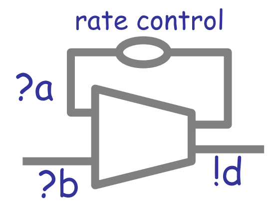


a=100 may or may not happen

Changing the OpAmp Gain

An OpAmp provides "infinite" differential amplification, but a stable finite amplification can be obtained by a feedback loop with a load splitter (the *follower* is a special case of that, which gives gain 1). The equivalent here is simply changing the rate on the feedback link.

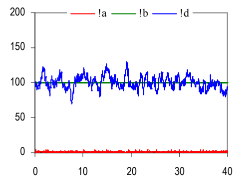




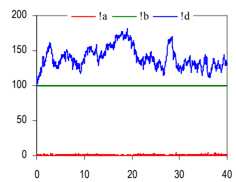
Empirical law:

[d] = [b]/rate(a)

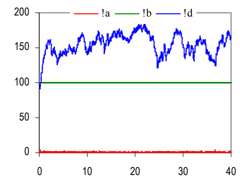
```
directive sample 40.0 1000
directive plot !a; !b; !d

new a@1.0:chan new b@1.0:chan new d@1.0:chan

let Xor_hi_a(a:chan, b:chan, c:chan, d:chan) =
  do !c; Xor_hi_a(a,b,c,d) or ?b; Xor_lo_ab(a,b,c,d) or delay@1.0; Xor_lo_a(a,b,c,d)
and Xor_hi_b(a:chan, b:chan, c:chan, d:chan) =
  do !d; Xor_hi_b(a,b,c,d) or ?a; Xor_lo_ab(a,b,c,d) or delay@1.0; Xor_lo_b(a,b,c,d)
and Xor_lo_a(a:chan, b:chan, c:chan, d:chan) =
  do ?a; Xor_hi_a(a,b,c,d) or ?b; Xor_lo_ab(a,b,c,d)
and Xor_lo_b(a:chan, b:chan, c:chan, d:chan) =
  do ?b; Xor_hi_b(a,b,c,d) or ?a; Xor_lo_ab(a,b,c,d)
and Xor_lo_ab(a:chan, b:chan, c:chan, d:chan) =
  do delay@1.0; Xor_hi_a(a,b,c,d) or delay@1.0; Xor_hi_b(a,b,c,d)

run 100 of (Xor_lo_a(a,b,a,d) | Xor_lo_b(a,b,a,d))
run 100 of replicate !b
```



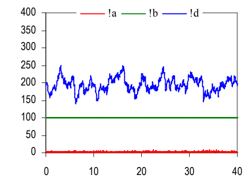
b=100
a@1.0
d gain 1.0
#OpAmp=200



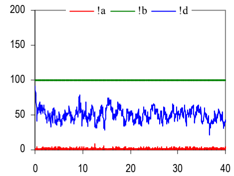
b=100
a@0.75
d gain 1.33
#OpAmp=200



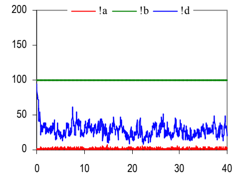
b=100
a@0.6
d gain 1.66
#OpAmp=200



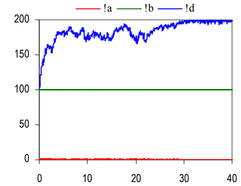
b=100
a@0.5
d gain 2.00
#OpAmp=400
(non saturated)



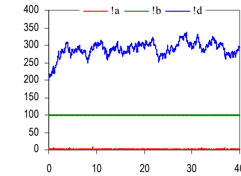
b=100
a@2.0
d gain 0.5
#OpAmp=200



b=100
a@4.0
d gain 0.25
#OpAmp=200



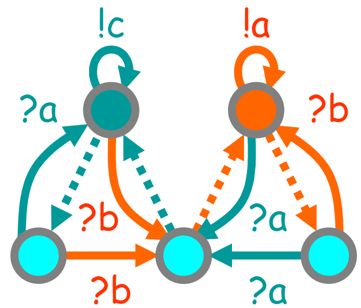
b=100
a@0.5
d gain 2.00
#OpAmp=200
(saturated)

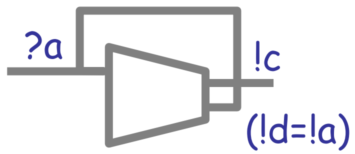


b=100
a@0.33
d gain 3.00
#OpAmp=400

Op Amp Inverting Configuration

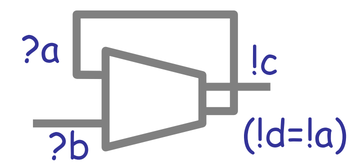
```
directive sample 40.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@1.0:chan

let Xor_hi_a(a:chan, b:chan, c:chan, d:chan) =
  do !c; Xor_hi_a(a,b,c,d) or ?b; Xor_lo_ab(a,b,c,d) or delay@1.0; Xor_lo_a(a,b,c,d)
and Xor_hi_b(a:chan, b:chan, c:chan, d:chan) =
  do !d; Xor_hi_b(a,b,c,d) or ?a; Xor_lo_ab(a,b,c,d) or delay@1.0; Xor_lo_b(a,b,c,d)
and Xor_lo_a(a:chan, b:chan, c:chan, d:chan) =
  do ?a; Xor_hi_a(a,b,c,d) or ?b; Xor_lo_ab(a,b,c,d)
and Xor_lo_b(a:chan, b:chan, c:chan, d:chan) =
  do ?b; Xor_hi_b(a,b,c,d) or ?a; Xor_lo_ab(a,b,c,d)
and Xor_lo_ab(a:chan, b:chan, c:chan, d:chan) =
  do delay@1.0; Xor_hi_a(a,b,c,d) or delay@1.0; Xor_hi_b(a,b,c,d)

run 100 of (Xor_lo_a(a,b,c,a) | Xor_lo_b(a,b,c,a))
run 1 of replicate !a
```





"Inverting Configuration"

c level depends on a and rate(a)
i.a. a signal is amplified according to rate(a)

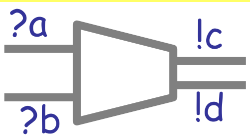


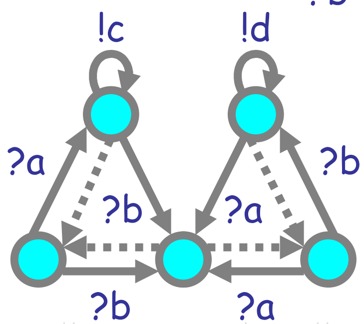
"Inverting Configuration"

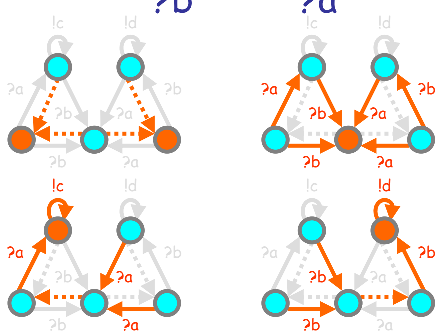
c = not b
a zero (ideally, if rate(a) fast enough)
rate(a) has no effect on c

An Xor but Not an Op Amp

c = A*(a - b)
d = A*(b - a)

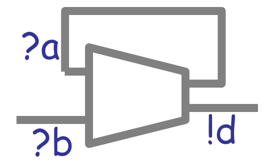


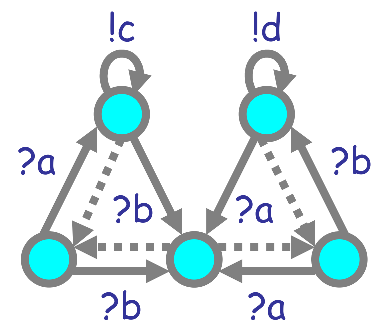


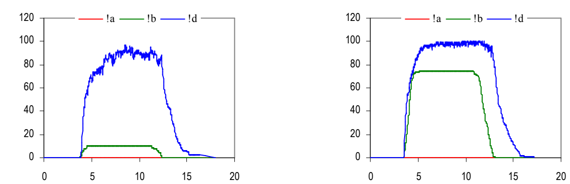


```
directive sample 20.0 1000
directive plot !a; !b; !c; !d

let Xor_hi_a(a:chan, b:chan, c:chan, d:chan) =
  do !c; Xor_hi_a(a,b,c,d) or ?b; Xor_lo_ab(a,b,c,d) or delay@1.0; Xor_lo_a(a,b,c,d)
and Xor_hi_b(a:chan, b:chan, c:chan, d:chan) =
  do !d; Xor_hi_b(a,b,c,d) or ?a; Xor_lo_ab(a,b,c,d) or delay@1.0; Xor_lo_b(a,b,c,d)
and Xor_lo_a(a:chan, b:chan, c:chan, d:chan) =
  do ?a; Xor_hi_a(a,b,c,d) or ?b; Xor_lo_ab(a,b,c,d)
and Xor_lo_b(a:chan, b:chan, c:chan, d:chan) =
  do ?b; Xor_hi_b(a,b,c,d) or ?a; Xor_lo_ab(a,b,c,d)
and Xor_lo_ab(a:chan, b:chan, c:chan, d:chan) =
  do delay@1.0; Xor_lo_a(a,b,c,d) or delay@1.0; Xor_lo_b(a,b,c,d)

new a@1.0:chan new b@1.0:chan new c@1.0:chan new d@1.0:chan

run 50 of (Xor_lo_a(a,b,c,d) | Xor_lo_b(a,b,c,d))

let clock(t:float, tick:chan) =      (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_a(tick:chan) = do !a; S_a(tick) or ?tick; ()
let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

run 10 of (new tick:chan run (clock(8.0,tick) | S_a(tick)))
run 10 of (new tick:chan run (clock(4.0,tick) | S_b(tick)))
```

Not a Follower







d≠b !!

```
directive sample 20.0 1000
directive plot !a; !b; !d

let Xor_hi_a(a:chan, b:chan, c:chan, d:chan) =
  do !c; Xor_hi_a(a,b,c,d) or ?b; Xor_lo_ab(a,b,c,d) or delay@1.0; Xor_lo_a(a,b,c,d)
and Xor_hi_b(a:chan, b:chan, c:chan, d:chan) =
  do !d; Xor_hi_b(a,b,c,d) or ?a; Xor_lo_ab(a,b,c,d) or delay@1.0; Xor_lo_b(a,b,c,d)
and Xor_lo_a(a:chan, b:chan, c:chan, d:chan) =
  do ?a; Xor_hi_a(a,b,c,d) or ?b; Xor_lo_ab(a,b,c,d)
and Xor_lo_b(a:chan, b:chan, c:chan, d:chan) =
  do ?b; Xor_hi_b(a,b,c,d) or ?a; Xor_lo_ab(a,b,c,d)
and Xor_lo_ab(a:chan, b:chan, c:chan, d:chan) =
  do delay@1.0; Xor_lo_a(a,b,c,d) or delay@1.0; Xor_lo_b(a,b,c,d)

new a@1.0:chan new b@1.0:chan new d@1.0:chan

run 50 of (Xor_lo_a(a,b,a,d) | Xor_lo_b(a,b,a,d))

let clock(t:float, tick:chan) =      (* sends a tick every t time *)
  (val ti = t/200.0 val d = 1.0/ti
   let step(n:int) =
     if n<=0 then !tick; clock(t, tick) else delay@d; step(n-1)
   run step(200))

let S_b(tick:chan) = ?tick; S_b1(tick)
and S_b1(tick:chan) = do !b; S_b1(tick) or ?tick; S_b2(tick)
and S_b2(tick:chan) = do !b; S_b2(tick) or ?tick; ()

run 10 of (new tick:chan run (clock(4.0,tick) | S_b(tick)))
```

Exercise (Open)

- Find the ODEs of some Xor or OpAmp configuration (e.g. Follower), and possibly derive some laws from them.

Summary

- Influence Diagrams
  - Don't trust them
- Polin Diagrams
  - An alternate influence-like notation for interacting automata
- Monopolin Circuits
  - Amplifiers
  - Inverters
  - Boolean Gates
  - OpAmp

Q?