Artificial Biochemistry

Combining Stochastic Collectives

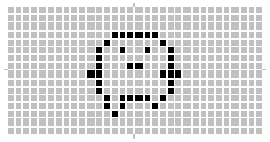
Luca Cardelli

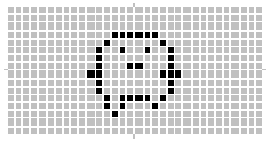
Microsoft Research

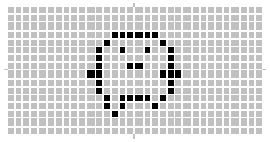
Newton Institute, Logic and Algorithms Program
2006-05-09

www.luca.demon.co.uk

Stochastic Collectives







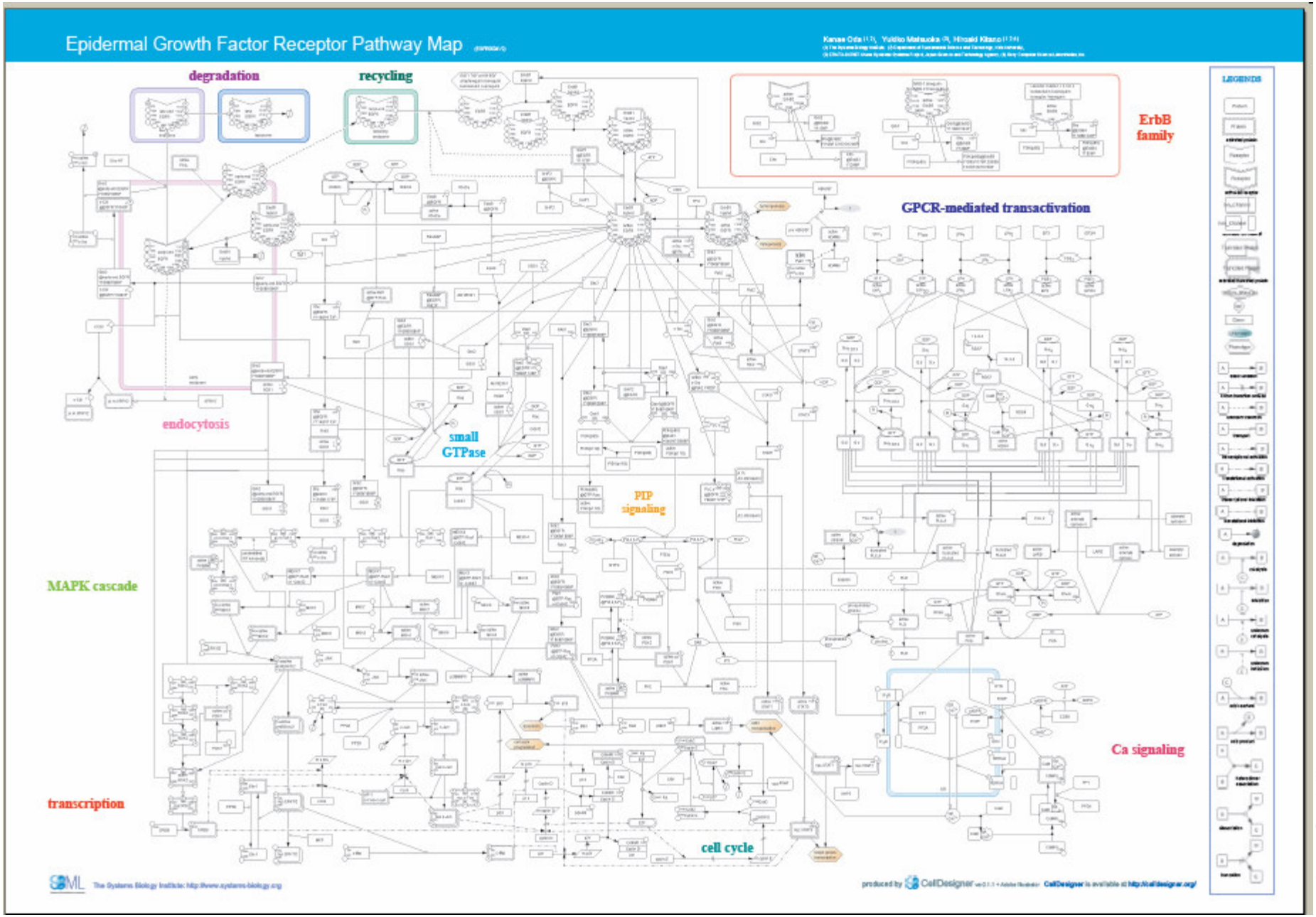
Stochastic Collectives

- "Collective":
  - A large set of interacting finite state automata:
    - Not quite language automata ("large set")
    - Not quite cellular automata ("interacting" but not on a grid)
    - Not quite process algebra ("finite state" and "collective")
    - Cf. "multi-agent systems" and "swarm intelligence"

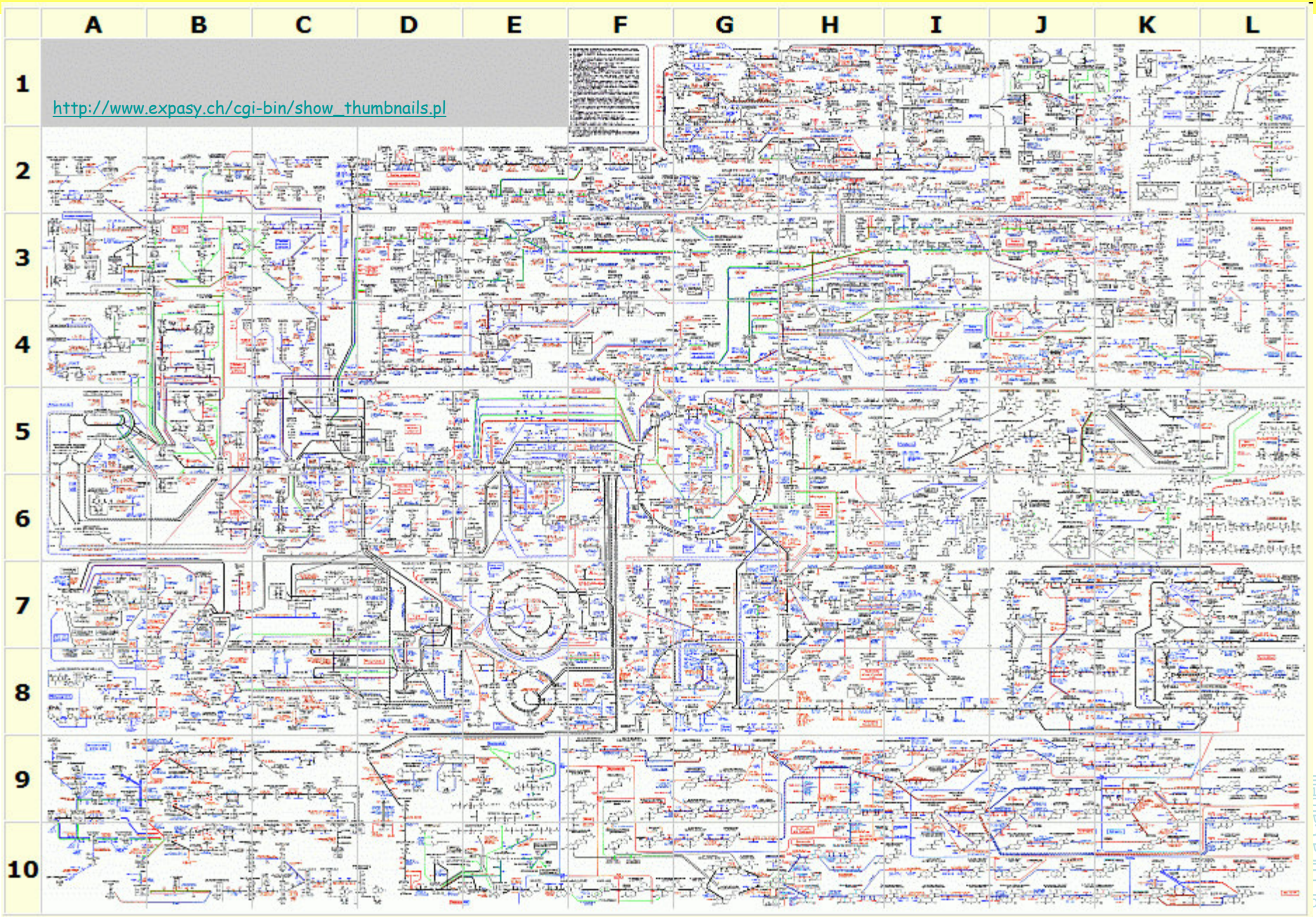
- "Stochastic":
  - Interactions have *rates*
    - Not quite discrete (hundreds or thousands of components)
    - Not quite continuous (non-trivial stochastic effects)
    - Not quite hybrid (no "switching" between regimes)

- Very much like biochemistry
  - Which is a large set of stochastically interacting molecules/proteins
  - Are proteins finite state and subject to automata-like transitions?
    - Let's say they are, at least because:
    - Much of the knowledge being accumulated in Systems Biology is described as state transition diagrams [Kitano].

State Transitions



Even More State Transitions



Reverse Engineering Nature

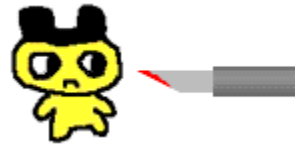
- That's what Systems Biology is up against
  - Exemplified by a technological analogy:

- Tamagotchi: a technological organism
  - Has inputs (buttons) and outputs (screen/sound)
  - It has state: happy or needy (or hungry, sick, dead…)
  - Has to be petted at a certain rate (or gets needy)
  - Each one has a slightly different behavior

How often do I have to exercise my Tamagotchi? Every Tamagotchi is different. However we do recommend exercising at least three times a day

- Reverse Engineering Tamagotchi
  - Running experiments that elucidate their behavior
  - Building models that explain the experiments

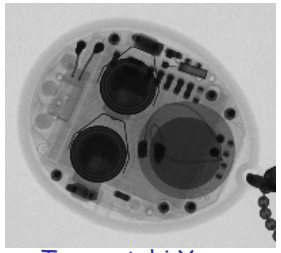
- Applications
  - Engineering: Can we build our own Tamagotchi?
  - Maintenance: Can we fix a broken Tamagotchi?



Understanding T.Nipponensis

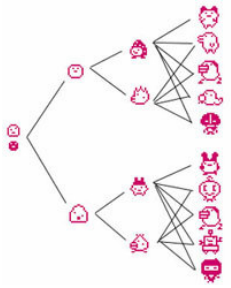
- Tamagotchi Nipponensis: a stochastic interactive automaton
  - 40 million sold worldwide; discontinued in 1998
  - Still found "in the wild" in Akihabara
  - New version in 2004: they communicate!

- Traditional scientific investigations fail
  - Design-driven understanding fails
    - We cannot read the manual (Japanese)
    - What does a Tamagotchi "compute"? What is its "purpose"?
    - Why does it have 3 buttons?
  - Mechanistic understanding fails
    - Few moving parts. Removing components mostly ineffective or "lethal"
    - The "tamagotchi folding problem" (sequence of manufacturing steps) is too hard and gives little insight on function
  - Behavioral understanding fails
    - Subjecting to extreme conditions reveals little and may void warranty
    - Does not answer consistently to individual stimuli, nor to sequences of stimuli
    - There are stochastic variations between individuals
  - Ecological understanding fails
    - Difficult to observe in its native environment (kids' hands)
    - Mass produced in little-understood automated factories
    - It evolved by competing with other products in the baffling Japanese market
  - Mathematical understanding fails
    - What differential equations does it obey? (Uh?)



Tamagotchi X-ray

Tamagotchi Surgery
http://necrobones.com/tamasurg/



A New Approach

- "Systems Technology" of T. Nipponensis
  - High-throughput experiments (get all the information you possibly can)
    - Decode the entire software and hardware
    - Take sequences of tamagotchi screen dumps under different conditions
    - Put 300 in a basket and shake them; make statistics of final state

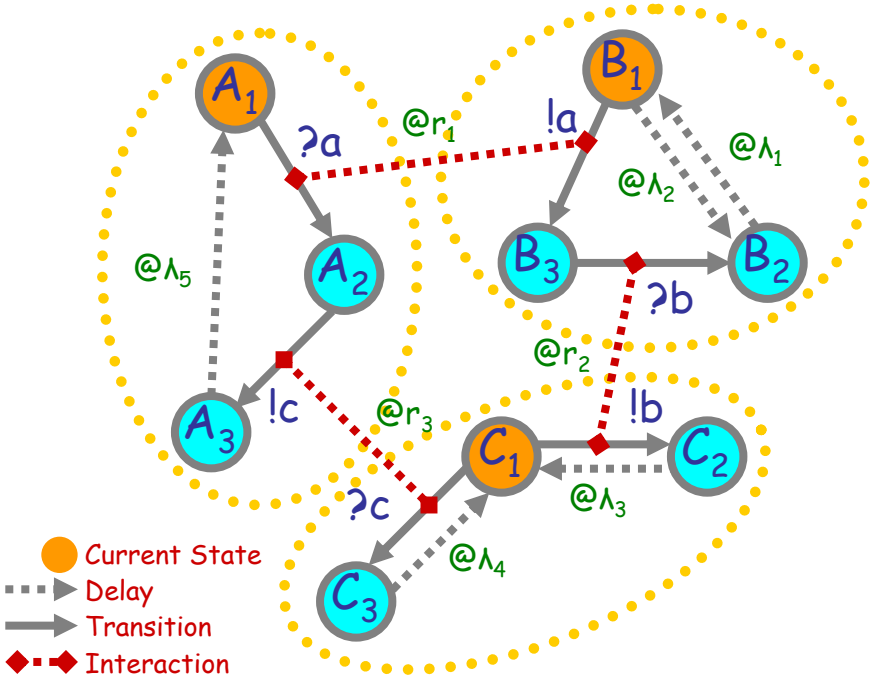
  - Modeling (organize all the information you got)
    - Ignore the "folding" (manufacturing) problem
    - Ignore materials (it's just something with buttons, display, and a *program*.)
    - Abstract until you find a conceptual model (ah-ha: it's a stochastic automaton).

- Do we understand what stochastic automata collectives can do?

Communicating Tamagotchi

Automata Collectives

Interacting Automata



Communicating automata: a graphical FSA-like notation for "finite state restriction-free π-calculus processes". *Interacting* automata do not even exchange values on communication.

The stochastic version has *rates* on communications, and delays.

$$
\begin{aligned}
&\text{new } a@r_1 \\
&\text{new } b@r_2 \\
&\text{new } c@r_3
\end{aligned}
\Big\}\ \text{Communication channels}
$$

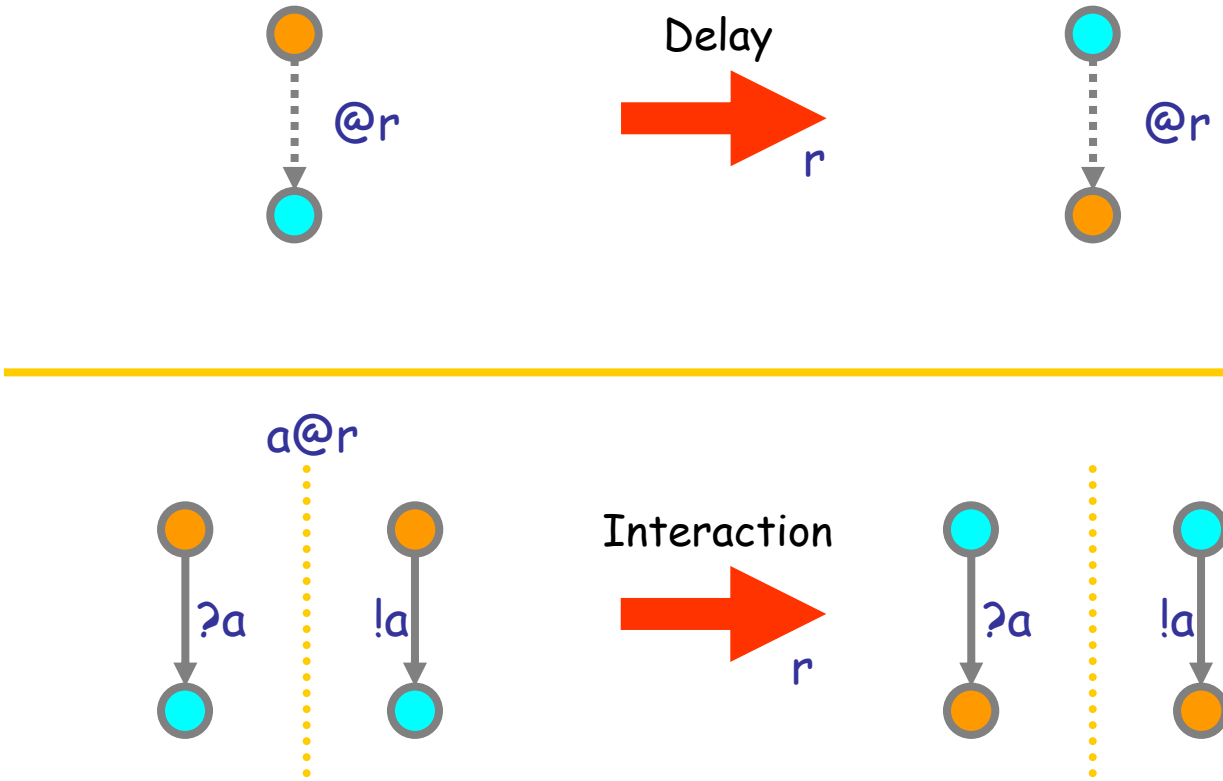
$$
\begin{aligned}
A_1 &= ?a;\ A_2 \\
A_2 &= !c;\ A_3 \\
A_3 &= @\lambda_5;\ A_1 \\
B_1 &= @\lambda_2;\ B_2 + !a;\ B_3 \\
B_2 &= @\lambda_1;\ B_1 \\
B_3 &= ?b;\ B_2 \\
C_1 &= !b;\ C_2 + ?c;\ C_3 \\
C_2 &= @\lambda_3;\ C_1 \\
C_3 &= @\lambda_4;\ C_2
\end{aligned}
\Big\}\ \text{Automata}
$$

$$
A_1 \mid B_1 \mid C_1 \Big\}\ \text{The system and initial state}
$$

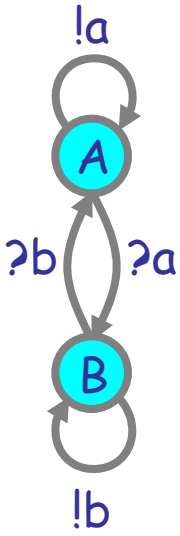
"Finite state" means: no composition or restriction inside recursion.

Analyzable by standard Markovian techniques, by first computing the "product automaton" to obtain the underlying finite Markov transition system. [Buchholz]

Interacting Automata Transition Rules



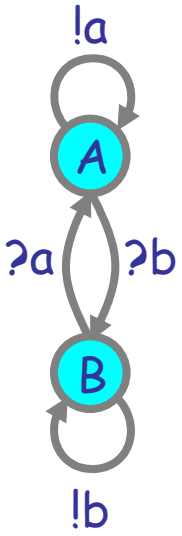
Groupies and Celebrities



Celebrity

(does not want to be like somebody else)

```
directive sample 0.1 1000
directive plot A(); B()

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

let A() = do !a; A() or ?a; B()
and B() = do !b; B() or ?b; A()

run 100 of (A() | B())
```

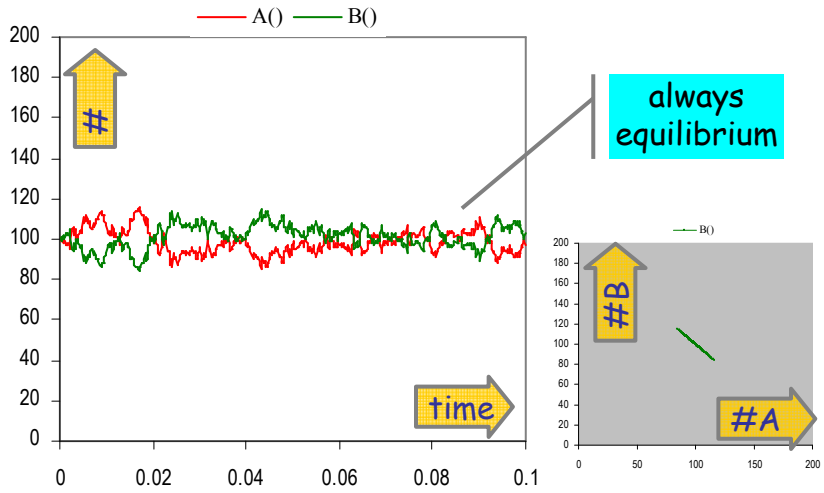


Groupie

(wants to be like somebody different)

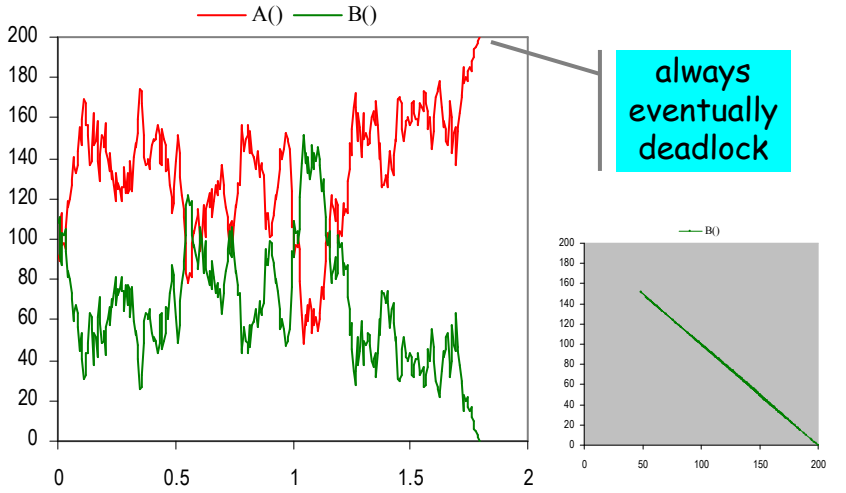
```
directive sample 5.0 1000
directive plot A(); B()

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

let A() = do !a; A() or ?b; B()
and B() = do !b; B() or ?a; A()

run 100 of (A() | B())
```

A stochastic collective of celebrities:



Stable because as soon as a A finds itself in the majority, it is more likely to find somebody in the same state, and hence change, so the majority is weakened.

A stochastic collective of groupies:



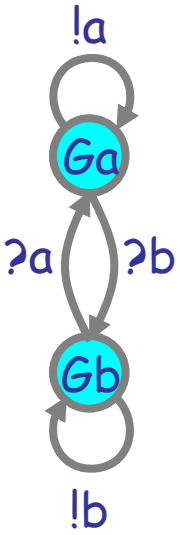
Unstable because within an A majority, an A has difficulty finding a B to emulate, but the few B's have plenty of A's to emulate, so the majority may switch to B. Leads to deadlock when everybody is in the same state and there is nobody different to emulate.

Both Together

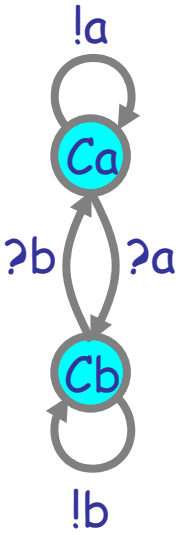
A tiny bit of "noise" can make a huge difference

A way to break the deadlocks: Groupies with just a few Celebrities

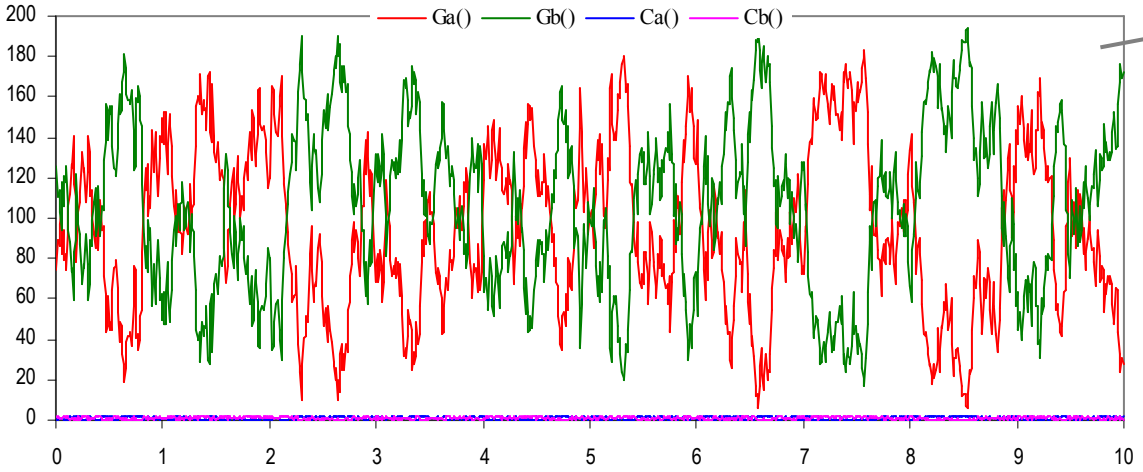
Many Groupies



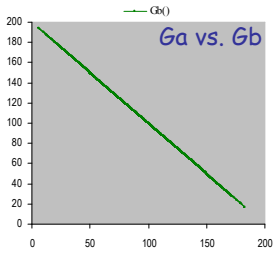
A few Celebrities




```
directive sample 10.0 1000
directive plot Ga(); Gb(); Ca(); Cb()

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

let Ca() = do !a; Ca() or ?a; Cb()
and Cb() = do !b; Cb() or ?b; Ca()

let Ga() = do !a; Ga() or ?b; Gb()
and Gb() = do !b; Gb() or ?a; Ga()

run   1 of (Ca() | Cb())
run 100 of (Ga() | Gb())
```

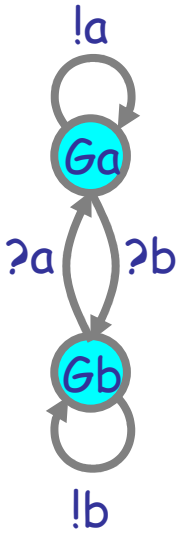


never deadlock

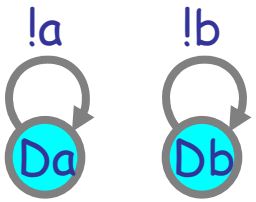


Doped Groupies

A similar way to break the deadlocks: destabilize the groupies by a small perturbation.

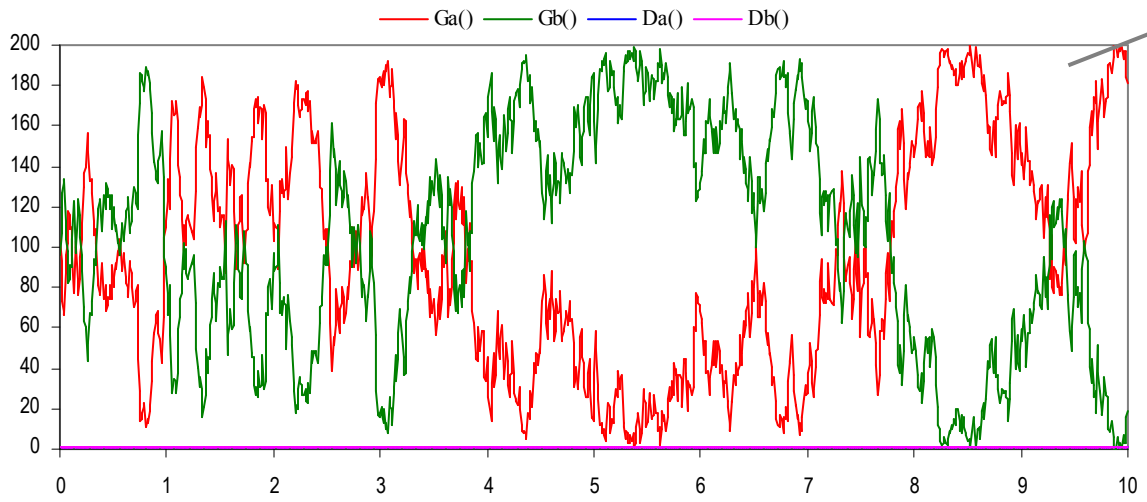


Groupie

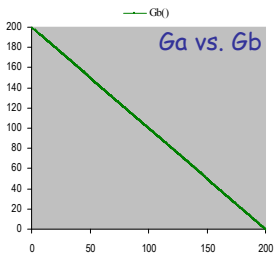


Doping[1]

```
directive sample 10.0 1000
directive plot Ga(); Gb(); Da(); Db()

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

let Ga() = do !a; Ga() or ?b; Gb()
and Gb() = do !b; Gb() or ?a; Ga()

let Da() = !a; Da()
and Db() = !b; Db()

run   1 of (Da() | Db())
run 100 of (Ga() | Gb())
```



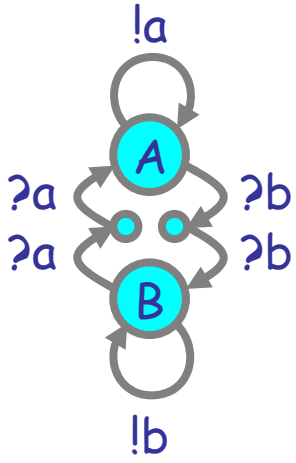
never deadlock

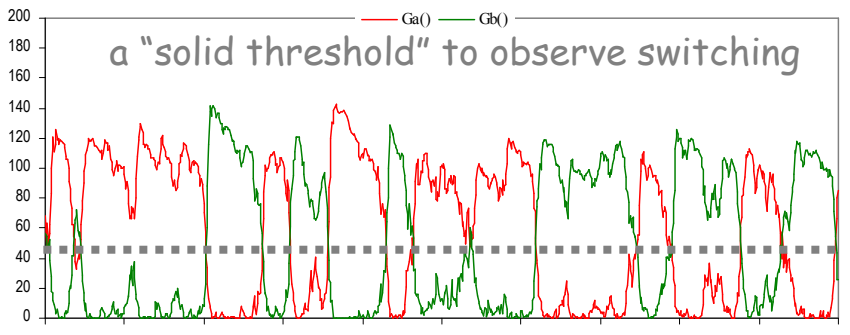


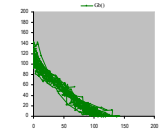
[1] A technical term in microelectronics

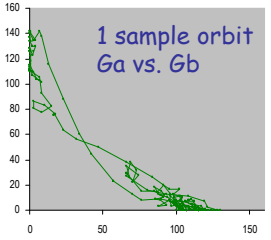
Hysteric Groupies

We can get more regular behavior from groupies if they "need more convincing", or "hysteresis" (history-dependence), to switch states.

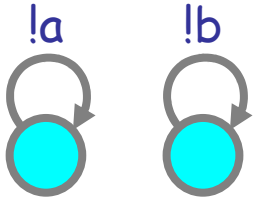






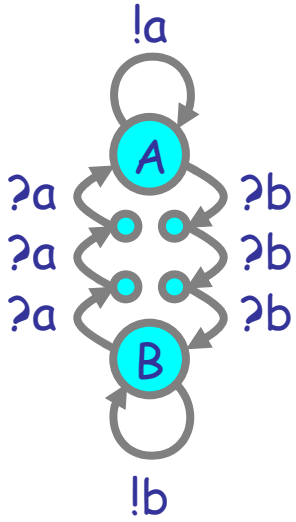


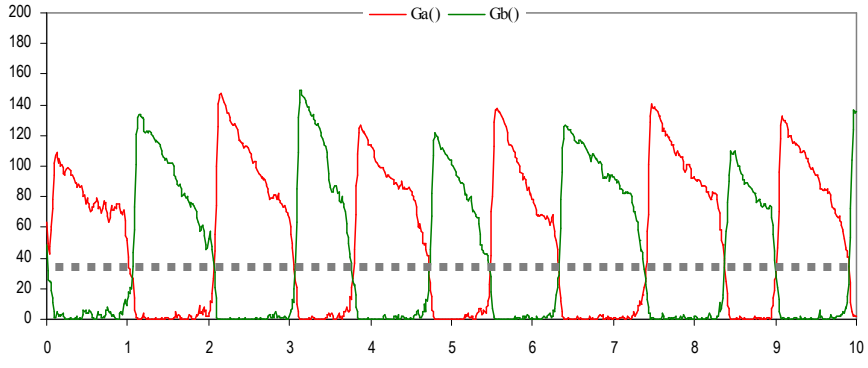

```
directive sample 10.0 1000
directive plot Ga(); Gb()

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

let Ga() = do !a; Ga() or ?b; ?b; Gb()
and Gb() = do !b; Gb() or ?a; ?a; Ga()

let Da() = !a; Da()
and Db() = !b; Db()

run 100 of (Ga() | Gb())
run   1 of (Da() | Db())
```

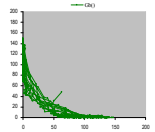


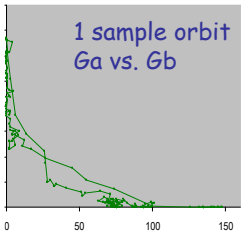
(Still with doping)

N.B.: It will not oscillate without doping (noise)



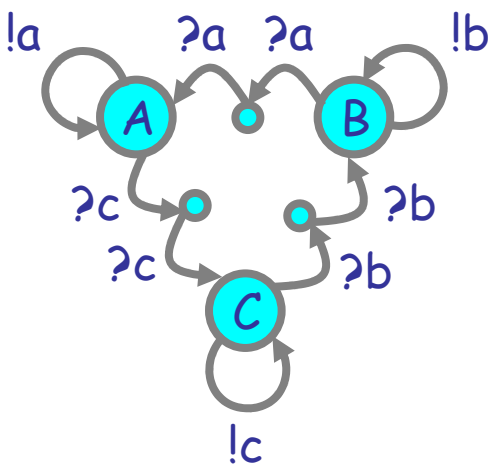


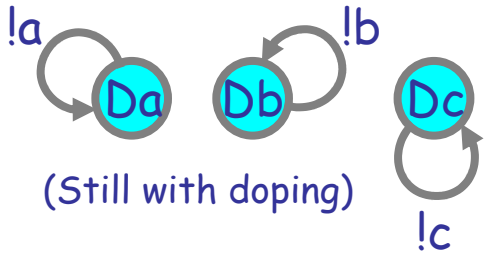





```
directive sample 10.0 1000
directive plot Ga(); Gb()

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

let Ga() = do !a; Ga() or ?b; ?b; ?b; Gb()
and Gb() = do !b; Gb() or ?a; ?a; ?a; Ga()

let Da() = !a; Da()
and Db() = !b; Db()

run 100 of (Ga() | Gb())
run   1 of (Da() | Db())
```

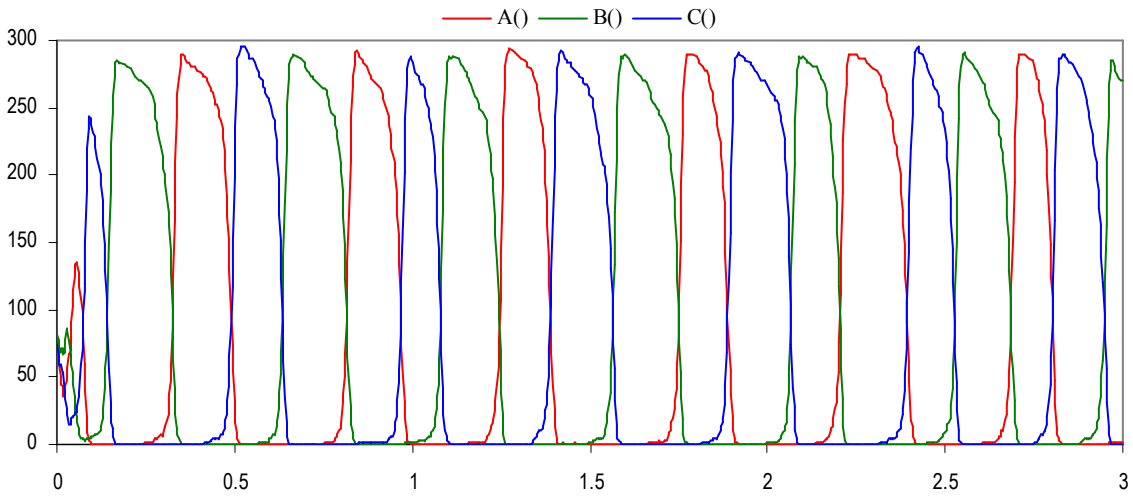
Hysteric 3-Way Groupies

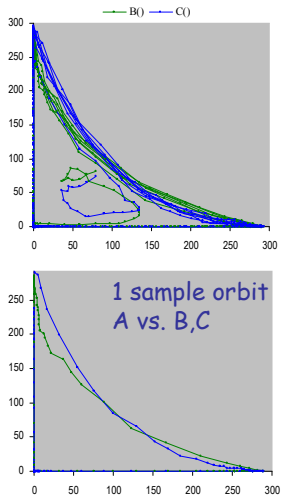




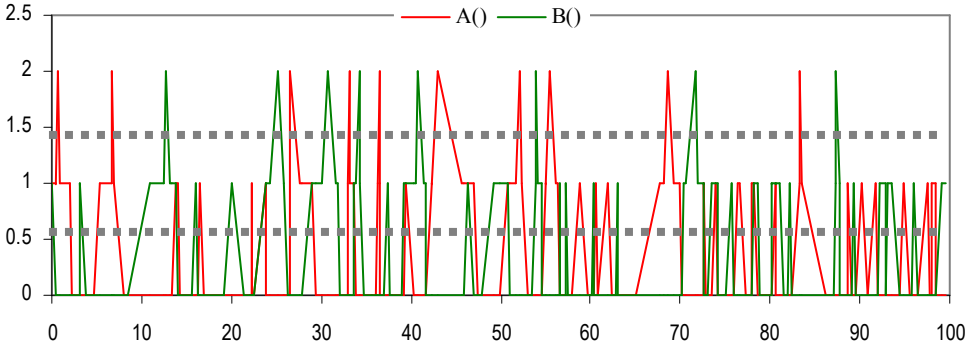

```
directive sample 3.0 1000
directive plot A(); B(); C()

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

let A() = do !a; A() or ?c; ?c; C()
and B() = do !b; B() or ?a; ?a; A()
and C() = do !c; C() or ?b; ?b; B()

let Da() = !a; Da()
and Db() = !b; Db()
and Dc() = !c; Dc()

run 100 of (A() | B() | C())
run 1 of (Da() | Db() | Dc())
```

N.B.: It will not oscillate without doping (noise)



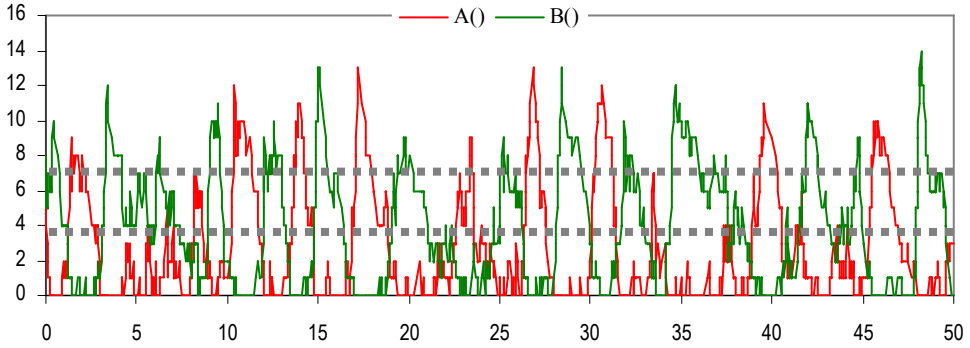


Oscillation as Emergence



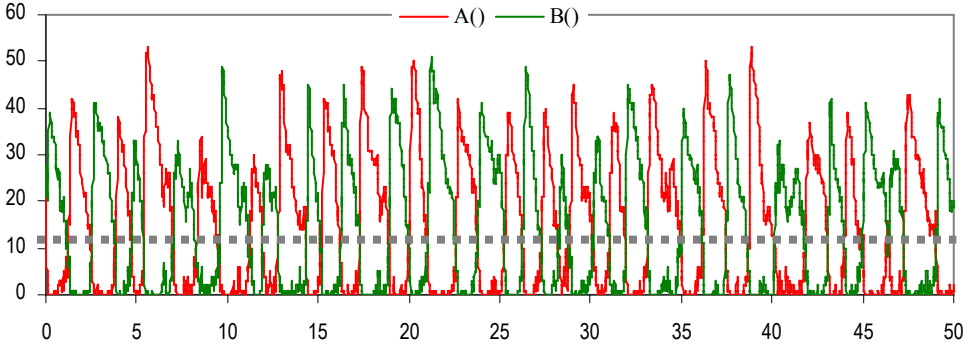
Just 2 of the hysteric groupies do not oscillate regularly at all!

Without changing the components, interesting properties emerge with a critical size of the population.



Nor 16…

Dotted lines indicate cross sections where one may look for evidence of alternation.

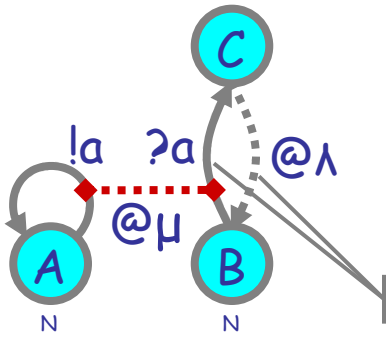


Pretty good with 64…

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

let A() = do !a; A() or ?b; ?b; ?b; B()
and B() = do !b; B() or ?a; ?a; ?a; A()

let As() = !a; As()
and Bs() = !b; Bs()

run 64 of (A() | B())
run  1 of (As() | Bs())
```

Collective Boolean Logic

The Strength of Populations

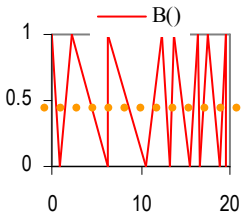


fight!

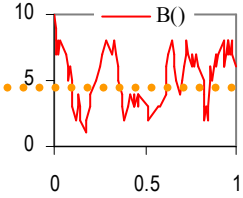
At size 2N, on a shared channel,
μ is N times stronger than λ:
interaction easily wins over delay.

```
directive sample 0.01 1000
directive plot B()

val lam = 1000.0
val mu = 1.0

new a@mu:chan
let A() = !a; A()
and B() = ?a; C()
and C() = delay@lam; B()

run 1000 of (A() | B())
```

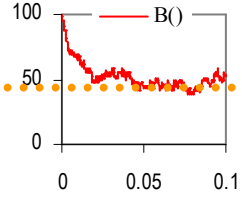
N=1
λ=1
μ=1



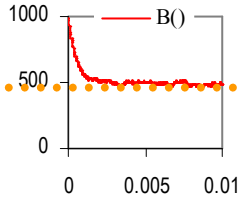
N=10
λ=10
μ=1



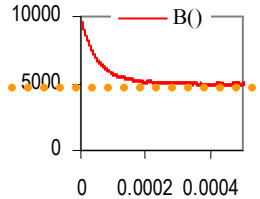
N=100
λ=100
μ=1



N=1000
λ=1000
μ=1



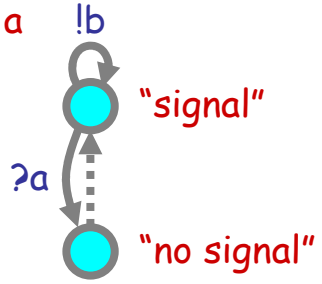
N=10000
λ=10000
μ=1



Equilibrium

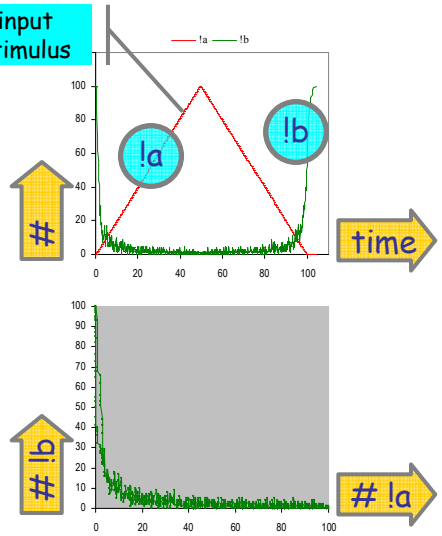
Boolean Inverter Collectives

b = not a



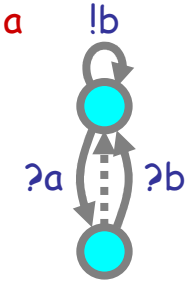
in presence of a, b goes low
in absence of a, b goes high

input stimulus

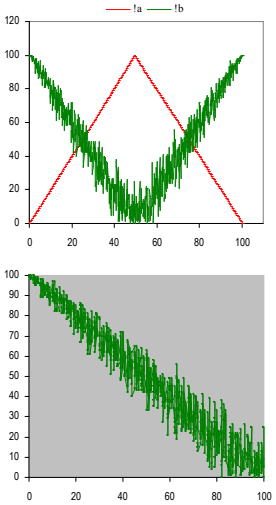



```
directive sample 110.0 1000
directive plot !a; !b

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

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

run 100 of Inv_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)))

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

b = not a

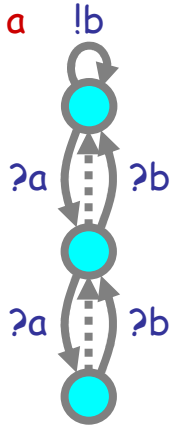


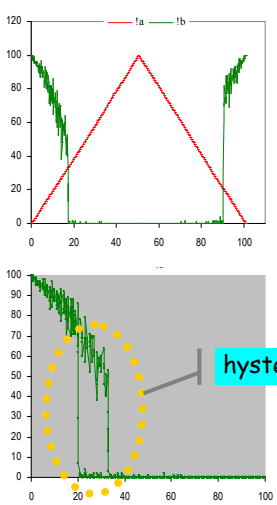
the high b state reinforces itself (as a population)



```
directive sample 110.0 1000
directive plot !a; !b

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

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

run 100 of Inv_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)))

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

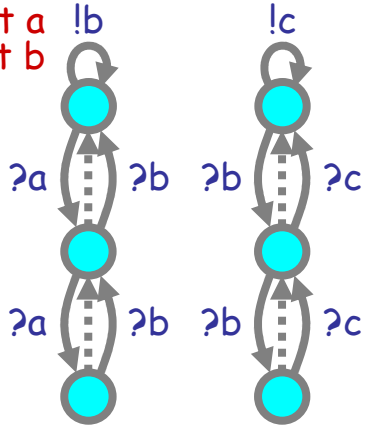
b = not a

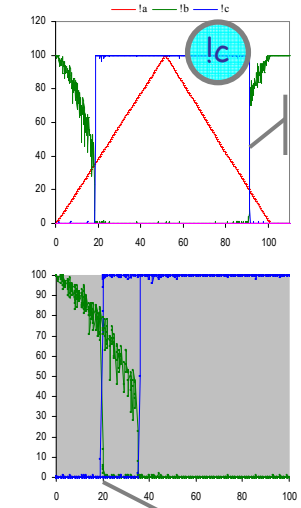





```
directive sample 110.0 1000
directive plot !a; !b

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

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

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)))

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

b = not a
c = not b

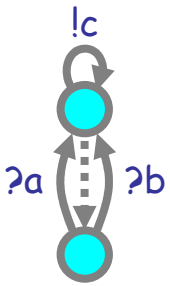




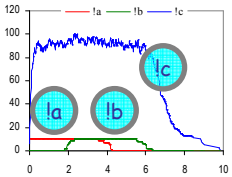

```
directive sample 110.0 1000
directive plot !a; !b; !c; !d

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

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

run 100 of (Inv2_hi(a,b) | Inv2_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 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)
```

Boolean Gate Collectives

c = a or b

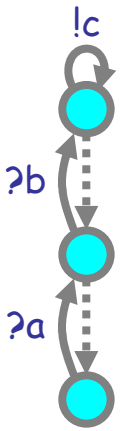


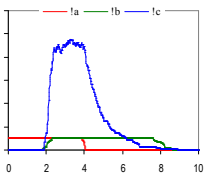
Inputs:
10 !a for 4t
2t; 10 !b for 4t




```
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)))
```

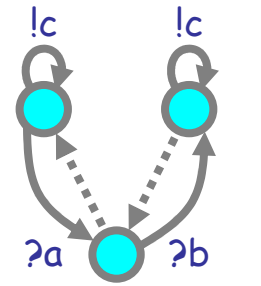
c = a and b

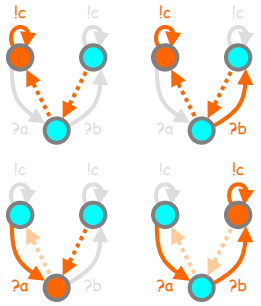


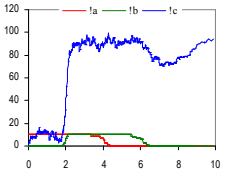


```
directive sample 10.0 1000
directive plot !a; !b; !c

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

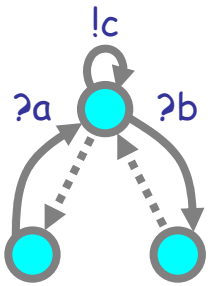


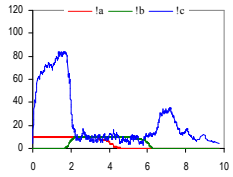




```
directive sample 10.0 1000
directive plot !a; !b; !c

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)))
```

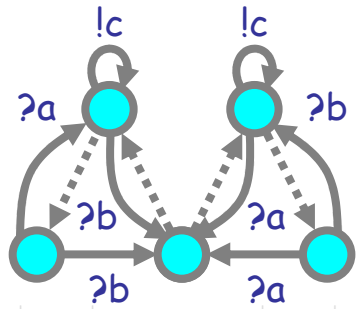
c = a unless b

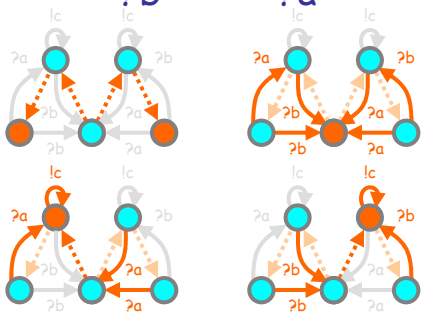


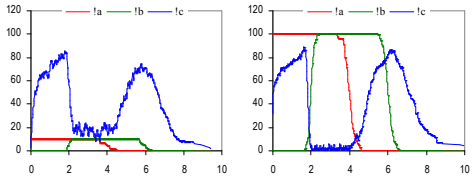


```
directive sample 10.0 1000
directive plot !a; !b; !c

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





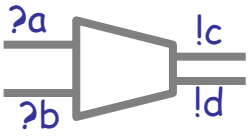


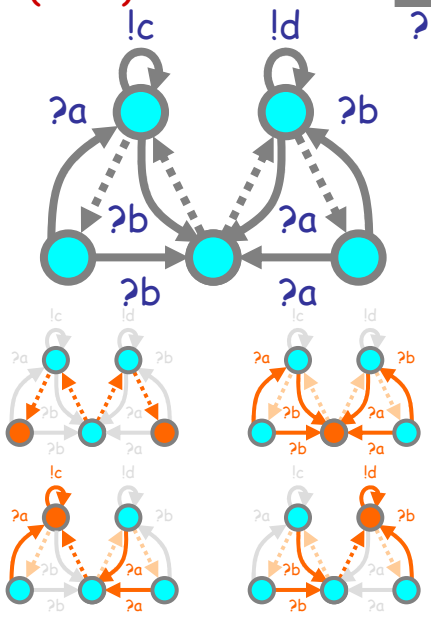
```
directive sample 10.0 1000
directive plot !a; !b; !c

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)))
```

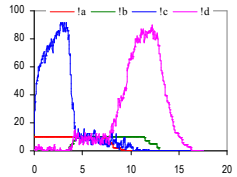
Collective Analog Devices

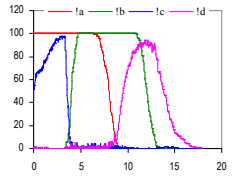
Xor as an Op Amp

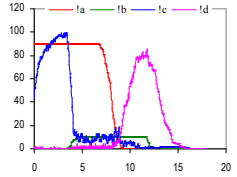
c = A*(a – b)
d = A*(b - a)









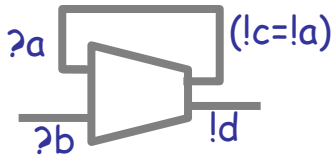



```
directive sample 20.0 1000
directive plot !a; !b; !c; !d

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_b1(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)))
```

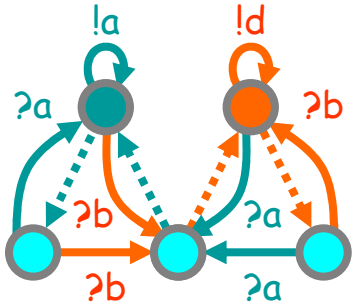
Follower (a standard OpAmp trick)

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

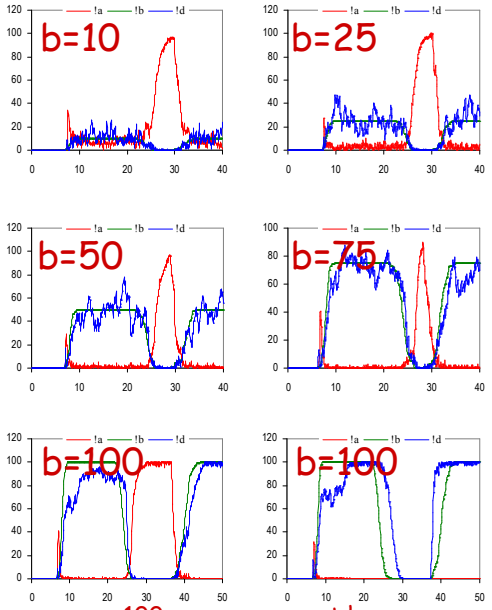
hence d=b



"Noninverting Configuration"



d=b analog response!!

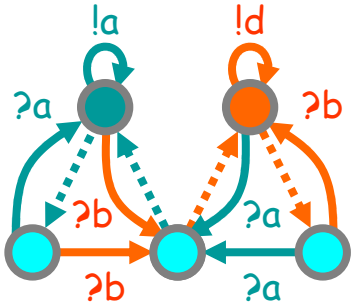


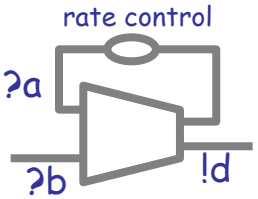
a=100 may or may not happen

```
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)))
```

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.



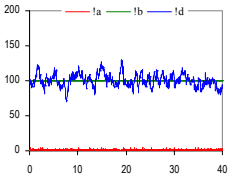


Empirical law:

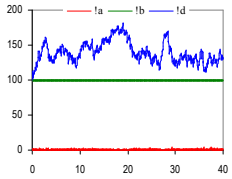
[d] = [b]/rate(a)

but *why?*

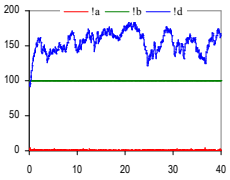
```
directive sample 40.0 1000
directive plot !a; !b; !d

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
```



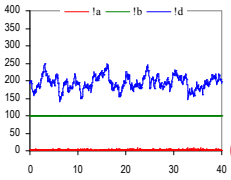
b=100
a@1.0
d gain 1.0
#OpAmp=200



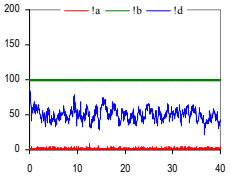
b=100
a@0.75
d gain 1.33
#OpAmp=200



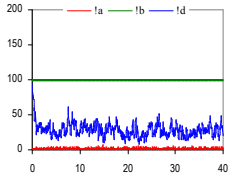
b=100
a@0.6
d gain 1.66
#OpAmp=200



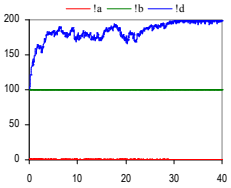
b=100
a@0.5
d gain 2.00
#OpAmp=400
(non saturated)



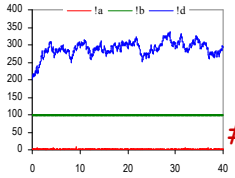
b=100
a@2.0
d gain 0.5
#OpAmp=200



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

Automata Polymers

Bidirectional Polymerization

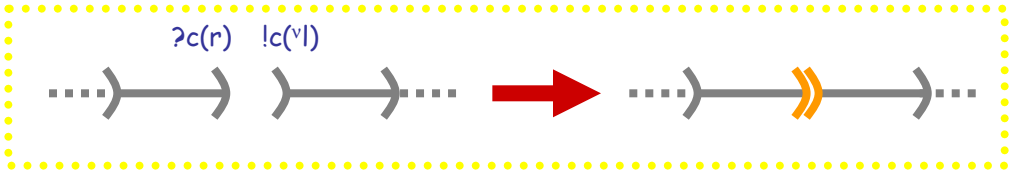
new c@μ new stop@1.0

A_{free} =
  (new rht@λ; !c(rht); A_{brht}(rht))
  + ?c(lft); A_{blft}(lft)

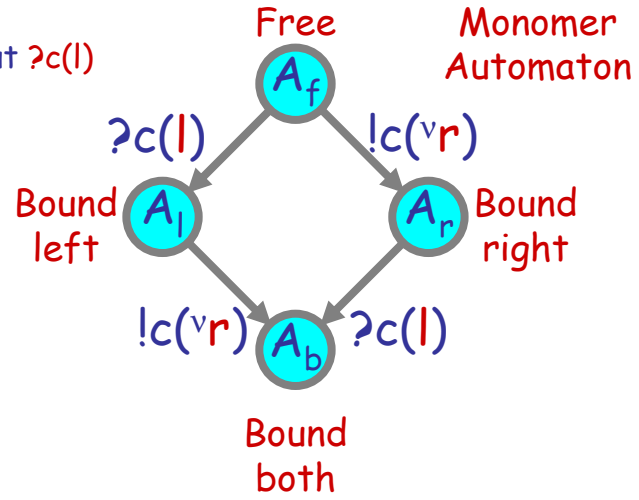
A_{blft}(lft) =
  (new rht@λ; !c(rht); A_{bound}(lft,rht))

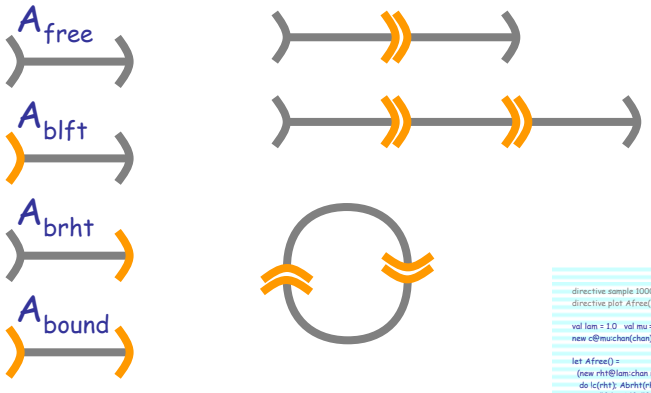
A_{brht}(rht) =
  ?c(lft); A_{bound}(lft,rht)

A_{bound}(lft,rht) = ?stop

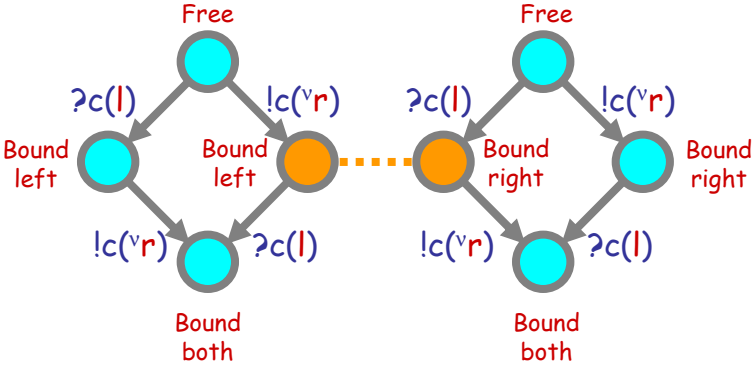


Communicating Automata
Bound output !c(ν r) and input ?c(l)
on automata transitions
to model complexation





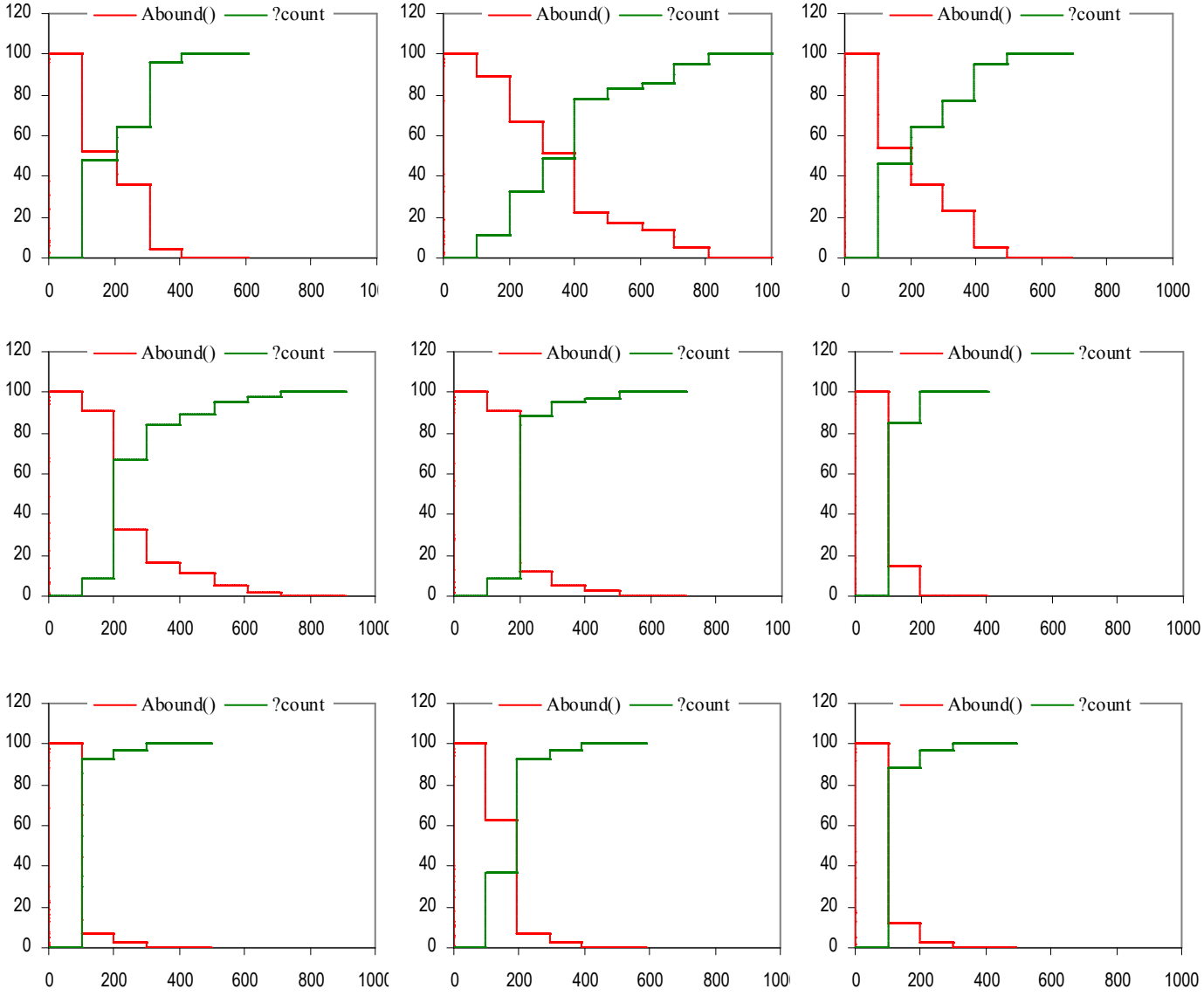

```
directive sample 10000.0
directive plot Afree(); Ablft(); Abrht(); Abound()

val lam = 1.0   val mu = 1.0
new c@mu:chan(chan)  new stop@1.0:chan

let Afree() =
 (new rht@lam:chan run
  do !c(rht); Abrht(rht)
  or ?c(lft); Ablft(lft))

and Ablft(lft:chan) =
 (new rht@lam:chan run
  !c(rht); Abound(lft,rht))

and Abrht(rht:chan) =
  ?c(lft); Abound(lft,rht)

and Abound(lft:chan, rht:chan) =
  ?stop

run (2 of Afree())
```



Bidirectional Polymerization

Circular Polymer Lengths

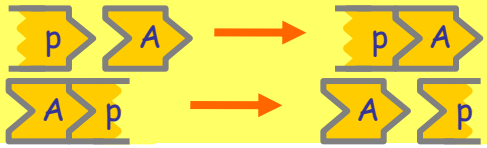
Scanning and counting the size of the circular polymers (by a cheap trick).
Polymer formation is complete within 10t; then a different polymer is scanned every 100t.




```
directive sample 1000.0
directive plot Abound(); ?count

type Link = chan(chan)
type Barb = chan

val lam = 1000.0 (* set high for better counting *)
val mu = 1.0
new c@mu:chan(Link)
new enter@lam:chan(Barb)
new count@lam:Barb

let Afree() =
  (new rht@lam:Link run
   do !c(rht); Abrht(rht)
   or ?c(lft); Ablft(lft))

and Ablft(lft:Link) =
  (new rht@lam:Link run
   !c(rht); Abound(lft,rht))

and Abrht(rht:Link) =
  ?c(lft); Abound(lft,rht)

and Abound(lft:Link, rht:Link) =
  do ?enter(barb); (?barb | !rht(barb))
  or ?lft(barb); (?barb | !rht(barb))
(* each Abound waits for a barb, exhibits it, and passes it to
   the right so we can plot number of Abound in a ring *)

let clock(t:float, tick:chan) =      (* sends a tick every t time *)
  (val ti = t/1000.0 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(1000))

new tick:chan
let Scan() = ?tick; !enter(count); Scan()

run 100 of Afree()
run (clock(100.0, tick) | Scan())
```

$100 \times A_{free}$, initially.

The height of each rising step is the size of a separate circular polymer. (Unbiased sample of nine consecutive runs.)



Actin-like
Poly/Depolymerization

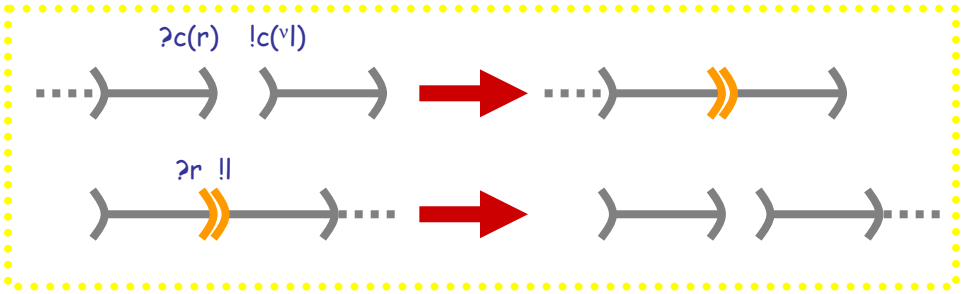
new c@μ

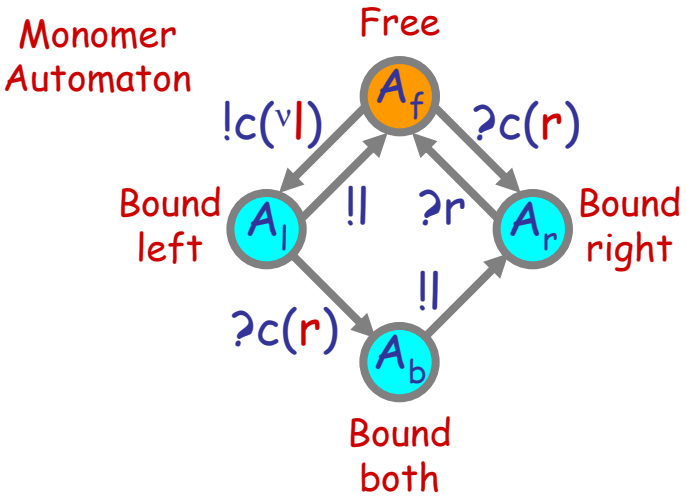
A_{free} =
  (new lft@λ; !c(lft); A_{blft}(lft)) +
  ?c(rht); A_{brht}(rht)

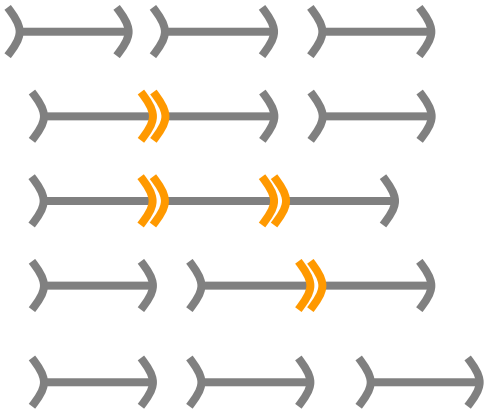
A_{blft}(lft) =
  !lft; A_{free} +
  ?c(rht); A_{bound}(lft,rht)

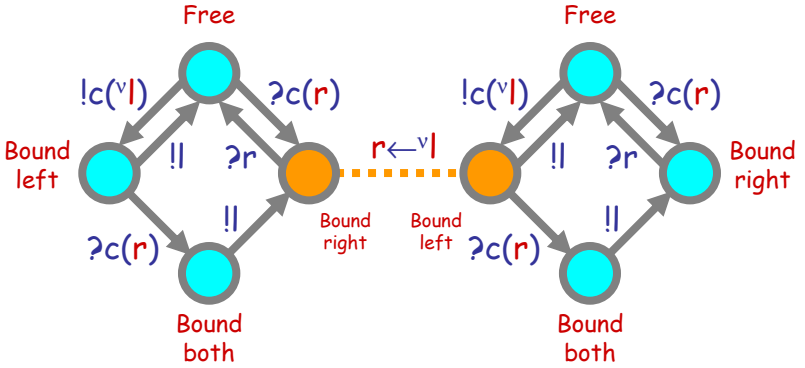
A_{brht}(rht) =
  ?rht; A_{free}

A_{bound}(lft,rht) =
  !lft; A_{brht}(rht)









Semantics of Collective Behavior

"Micromodels": Continuous Time Markov Chains

- The underlying semantics of stochastic π-calculus (and stochastic interacting automata). Well established in many ways.
  - Automata with rates on transitions.
- "The" correct semantics for chemistry, executable.
  - Gillespie stochastic simulation algorithm
- But does not give a good sense of "collective" properties.
  - Yes one can do simulation.
  - Yes one can do program analysis.
  - Yes one can do modelchecking.
  - But somewhat lacking in "predictive power".

"Macromodels": Ordinary Differential Equations

- Micromodels have lots of advantages
  - Compositional, compact, mechanistic, etc.

- But they always ask:
  - "Yes, but how does you automata model relate to the 75 ODE models in the literature?"

- From processes/automata to ODEs directly:
  - *In principle*: just write down the Rate Equation:
    - Determine the set of all possible *states* S of each process.
    - Determine the rates of the transitions between such states.
    - Let [S] be the "number of processes in state S" as a function of time.
    - Define for each state S:

      $[S]^{\bullet}$ = (rate of change of the number of processes in state S)

      Cumulative rate of transitions from any state S' to state S, times [S'],
      minus cumulative rate of transitions from S to any state S", times [S].

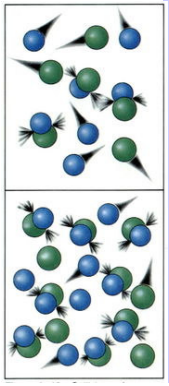
  - Intuitive (rate = inflow minus outflow), but often clumsy to write down precisely.

- But why go to the trouble?
  - If we first convert processes to chemical reactions, then we can convert to ODEs by standard means!

Macromodel of Interaction

Law of Mass Interaction

The speed of interaction† is proportional to the number of *possible interactions*.



Chemical Law of Mass Action

http://en.wikipedia.org/wiki/Chemical_kinetics

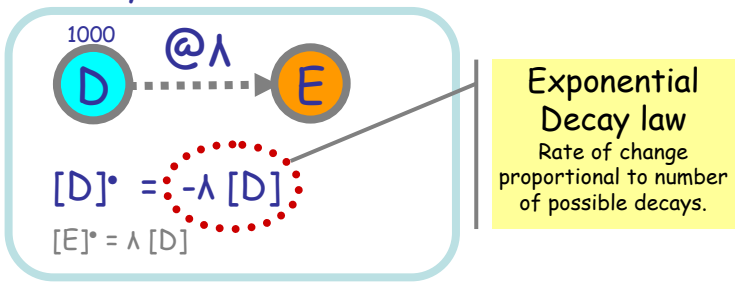
The **speed** of a chemical reaction is proportional to the **activity** of the reacting substances.

Activity = concentration, for well-stirred aqueous medium

Concentration = number of moles per liter of solution

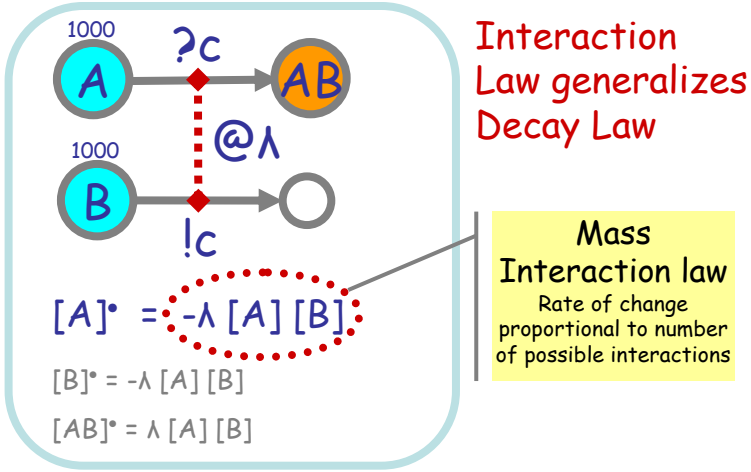
Mole = 6.022141×10^{23} particles

Decay



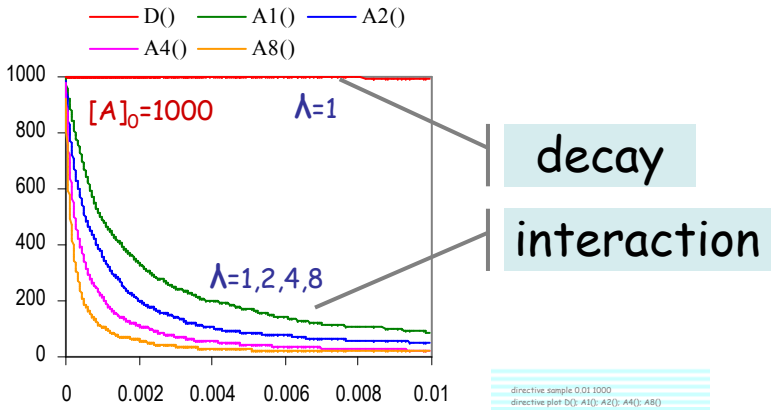
Exponential Decay law
Rate of change proportional to number of possible decays.

Mass interaction



Interaction Law generalizes Decay Law

Mass Interaction law
Rate of change proportional to number of possible interactions




```
directive sample 0.01 1000
directive plot D(); A1(); A2(); A4(); A8()

new c1@1.0: chan()  new c2@2.0: chan()
new c4@4.0: chan()  new c8@8.0: chan()

let D() = delay@1.0
let A1() = ?c1 and B1() = !c1
let A2() = ?c2 and B2() = !c2
let A4() = ?c4 and B4() = !c4
let A8() = ?c8 and B8() = !c8

run 1000 of (D() | A1() | B1() | A2()
| B2() | A4() | B4() | A8() | B8())
```

† speed of interaction (formally definable)
= number of interactions over time
not proportional to the number of interacting processes!

[P] is the number of processes P (this is informal; it is only meaningful for a set of processes offering a given action, but a set of such processes can be counted and plotted)

Possible Interactions

The speed of interaction is proportional to the number of possible interactions.
But a process cannot interact with itself.

Assume each process P is in restricted-sum-normal-form. For each channel x:

In(x,P) = Num of active ?x in P

Out(x,P) = Num of active !x in P

Mix(x,P) = In(x,P)*Out(x,P)

#interactions that cannot happen in a given summation P

In(x) = Sum P of In(x,P)

Out(x) = Sum P of Out(x,P)

Mix(x) = Sum P of Mix(x,P)

total #interactions that cannot happen

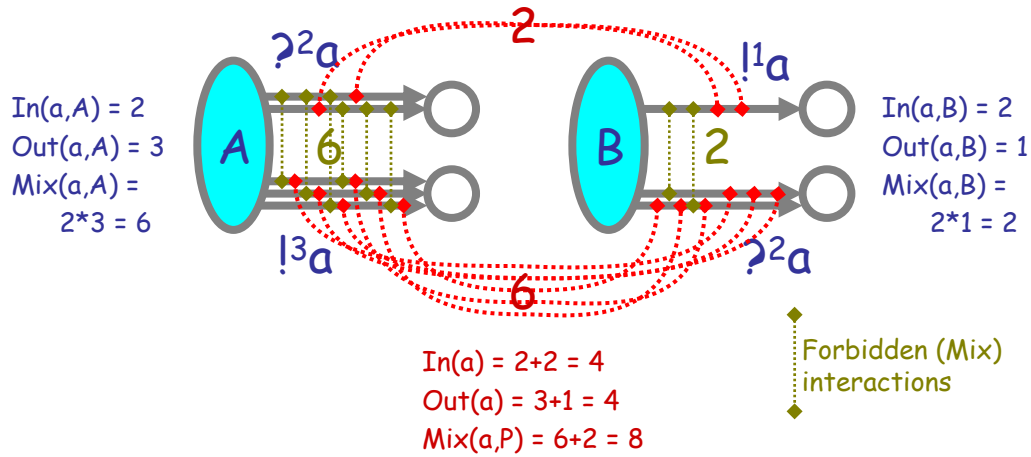
The global Activity on channel x:

Act(x) = (In(x)*Out(x))-Mix(x)

total cross product of inputs and outputs minus total #interactions that cannot happen

The global speed of interaction on a channel x:

speed(x) = Act(x)*rate(x)



Act(a) = (In(a) * Out(a)) – Mix(a) = 4*4 – 8 = 8

speed(a) = Act(a)*rate(a) = 8*rate(a)

From Chemistry to ODEs

Chemical Reactions

$A \rightarrow^{r} B_1 + \ldots + B_n$	Degradation	$[A]^{\bullet} = -r[A]$	Exponential Decay
$A_1 + A_2 \rightarrow^{r} B_1 + \ldots + B_n$	Asymmetric Collision	$[A_i]^{\bullet} = -r[A_1][A_2]$	Mass Action Law
$A + A \rightarrow^{r} B_1 + \ldots + B_n$	Symmetric Collision	$[A]^{\bullet} = -r[A]([A]-1)$	Mass Action Law

(assuming $A \neq B_i \neq A_j$ for all i,j)

No other reactions!

JOURNAL OF CHEMICAL PHYSICS VOLUME 113, NUMBER 1

The chemical Langevin equation

Daniel T. Gillespie[a)]
Research Department, Code 4T4100D, Naval Air Warfare Center, China Lake, California 93555

Genuinely *trimolecular* reactions do not physically occur in dilute fluids with any appreciable frequency. *Apparently* trimolecular reactions in a fluid are usually the combined result of two bimolecular reactions and one monomolecular reaction, and involve an additional short-lived species.

Chapter IV: Chemical Kinetics
[David A. Reckhow , CEE 572 Course]

... reactions may be either elementary or non-elementary. Elementary reactions are those reactions that occur exactly as they are written, without any intermediate steps. These reactions almost always involve just one or two reactants. ... Non-elementary reactions involve a series of two or more elementary reactions. Many complex environmental reactions are non-elementary. In general, reactions with an overall reaction order greater than two, or reactions with some non-integer reaction order are non-elementary.

THE COLLISION THEORY OF REACTION RATES
www.chemguide.co.uk

The chances of all this happening if your reaction needed a collision involving more than 2 particles are remote. All three (or more) particles would have to arrive at exactly the same point in space at the same time, with everything lined up exactly right, and having enough energy to react. That's not likely to happen very often!

Trimolecular reactions:

$A + B + C \rightarrow^{r} D$

the measured "r" is an (imperfect) aggregate of e.g.:

$A + B \leftrightarrow AB$

$AB + C \rightarrow D$

Enzymatic reactions:

$S \xrightarrow{E} {}^{r} P$

the "r" is given by Michaelis-Menten (approximated steady-state) laws:

$E + S \leftrightarrow ES$

$ES \rightarrow P + E$

From Reactions to ODEs

CAVEAT: A deterministic *approximation* of a stochastic system (i.e. possibly *dead wrong*)

v_1: $A+B \xrightarrow{k_1} C+C$

v_2: $A+C \xrightarrow{k_2} D$

v_3: $C \xrightarrow{k_3} E+F$

v_4: $F+F \xrightarrow{k_4} B$

Write the coefficients by columns

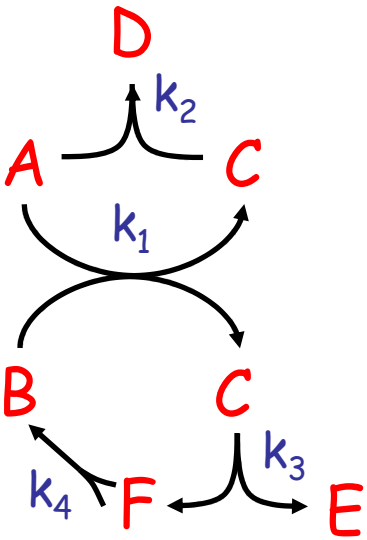
reactions

species

N	v_1	v_2	v_3	v_4
A	-1	-1		
B	-1			1
C	2	-1	-1	
D		1		
E			1	
F			1	-2

X

Stoichiometric Matrix



Quantity changes

Stoichiometric matrix

Rate laws

$$[X]^{\bullet} = N \cdot I$$

Read the concentration changes from the rows

$$[A]^{\bullet} = -I_1 - I_2$$
$$[B]^{\bullet} = -I_1 + I_4$$
$$[C]^{\bullet} = 2I_1 - I_2 - I_3$$
$$[D]^{\bullet} = I_2$$
$$[E]^{\bullet} = I_3$$
$$[F]^{\bullet} = I_3 - 2I_4$$

E.g. $[A]^{\bullet} = -k_1[A][B] - k_2[A][C]$

Set a rate law for each reaction
(Degradation/Asymmetric/Symmetric)

I	
I_1	$k_1[A][B]$
I_2	$k_2[A][C]$
I_3	$k_3[C]$
I_4	$k_4[F]([F]-1)/2$

X: chemical species
[-]: quantity of molecules
I: rate laws
k: kinetic parameters
N: stoichiometric matrix

From Processes to Chemistry

Chemical Ground Form (CGF)

$E ::= X_1=M_1, \ldots, X_n=M_n$ Definitions $(n\geq 0)$

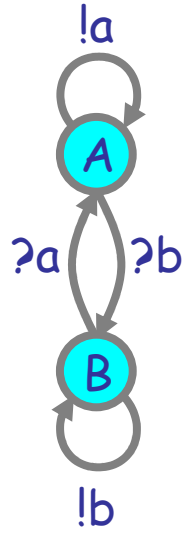
$M ::= \pi_1;P_1 \oplus \ldots \oplus \pi_n;P_n$ Molecules $(n\geq 0)$

$P ::= X_1 \mid \ldots \mid X_n$ Solutions $(n\geq 0)$

$\pi ::= \tau_r \;\; ?n_{(r)} \;\; !n_{(r)}$ Interactions (delay, input, output)

(To translate chemistry back to processes we need a bit more than simple automata: we may have "+" on the right of $\rightarrow$, that is we may need "|" after π.)

$\oplus$ is stochastic choice (vs. + for chemical reactions)
0 is the null solution ($P|0 = 0|P = P$)
and null molecule ($M\oplus 0 = 0\oplus M = M$) ($\tau_0;P = 0$)
X_i are distinct in E
Each name n is assigned a fixed rate r: $n_{(r)}$



Ex: interacting automata
(which are CGFs using "|" only in Init):

$A = !a;A \oplus ?b;B$ — Automaton in state A

$B = !b;B \oplus ?a;A$ — Automaton in state B

$Init = \tau_\infty;(A|A|B|B)$ — Initial conditions: 2A and 2B

CGF to Chemistry

$E ::= X_1=M_1, \ldots, X_n=M_n$	Definitions	$(n \geq 0)$
$M ::= \pi_1;P_1 \oplus \ldots \oplus \pi_n;P_n$	Molecules	$(n \geq 0)$
$P ::= X_1 \mid \ldots \mid X_n$	Solutions	$(n \geq 0)$
$\pi ::= \tau_r \quad ?n \quad !n$	Interactions	

Each X in E is seen as a separate *species*.

Chemical system for E: (N.B.: $\{\ldots\}^m$ is a multiset, and P is P with all the | changed to +)

$$Ch_G(E) := \{(X \to^r P) \text{ s.t. } (X \equiv \tau_r;P \oplus \ldots) \in E\}^m$$
$$\cup^m \{(X + Y \to^r P + Q) \text{ s.t. } X \neq Y, \langle (X \equiv ?n_{(r)};P \oplus \ldots),(Y \equiv !n_{(r)};Q \oplus \ldots)\rangle \in E^2\}^m$$
$$\cup^m \{(X + X \to^{2r} P + Q) \text{ s.t. } (X \equiv ?n_{(r)};P \oplus \ldots \equiv !n_{(r)};Q \oplus \ldots)\rangle \in E\}^m$$

(Note on computing the multisets)

A multiset $M \in$ Multiset(S), where S is a set with equality, is a total function S→Nat, which may also be written as a finite enumeration with repetitions: $\{\ldots\}^m$.

Multiset binary union is the function $\cup^m(M,M') = \backslash s.\ M(s)+M'(s)$.

The shorthand

$$\{(X \rightarrow^r P) \text{ s.t. } (X \equiv \tau_r;P \oplus \ldots) \in E\}^m$$

is defined as the following finite union of singleton multisets:

$$\cup^m\{(X=\pi_1;P_1\oplus\ldots\oplus\pi_n;P_n)\in E\}$$
$$\text{of}(\cup^m\{i \text{ s.t. } \pi_i=\tau_r\}$$
$$\text{of } \{(X \rightarrow^r P_i)\}^m)$$

i.e. "for each $(X=\pi_1;P_1\oplus\ldots\oplus\pi_n;P_n)\in E$ and for each i such that $\pi_i=\tau_r$, return a copy of $(X \rightarrow^r P_i)$".

The shorthand

$$\{(X + Y \rightarrow^r P + Q) \text{ s.t. } X\neq Y, \langle(X \equiv ?n_{(r)};P \oplus \ldots),(Y \equiv !n_{(r)};Q \oplus \ldots)\rangle \in E^2\}^m$$

is defined as the following finite union of singleton multisets:

$$\cup^m \langle(X=\pi_1;P_1\oplus\ldots\oplus\pi_n;P_n), (Y=\rho_1;Q_1\oplus\ldots\oplus\rho_m;Q_m)\rangle\in E^2 \text{ with } X\neq Y$$
$$\text{of } \cup^m\{\langle i,j\rangle \text{ s.t. } \pi_i=?n_{(r)},\ \rho_j=!n_{(r)}\}$$
$$\text{of } \{(X + Y \rightarrow^r P_i + Q_j)\}^m)$$

Example

E:

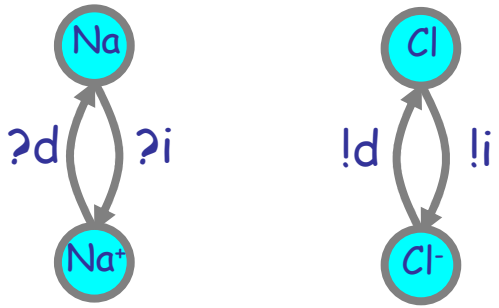
$Na = ?i;\ Na^+$
$Cl = !i;\ Cl^-$
$Na^+ = ?d;\ Na$
$Cl^- = !d;\ Cl$
$Init = \tau_\infty;\ (Na \mid Na \mid Cl \mid Cl)$

$Ch_G(E)$:

$Na + Cl \rightarrow^{\rho(i)} Na^+ + Cl^-$
$Na^+ + Cl^- \rightarrow^{\rho(d)} Na + Cl$
$Init \rightarrow^{\infty} Na + Na + Cl + Cl$

$\rho_E(n_{(r)}) = r$

$$Ch_G(E) := \{(X \rightarrow^r P) \text{ s.t. } (X \equiv \tau_r;P \oplus \ldots) \in E\}^m \cup^m \{(X + Y \rightarrow^r P + Q) \text{ s.t. } X \neq Y, \langle (X \equiv ?n_{(r)};P \oplus \ldots),(Y \equiv !n_{(r)};Q \oplus \ldots)\rangle \in E^2\}^m \cup^m \ldots$$



Subtler Examples

These are not *finite state systems*, but *finite species systems* are ok!

E:

$X = \tau_r;(X \mid X)$

C(E):

$X \rightarrow^{r} X + X$

Unbounded state,
but only 1 species.
No problem!

Multisets:

The same interaction can occur multiple times and must be taken into account:

E:

$A = ?n;B \oplus ?n;B$

$C = !n;D$

C(E):

$A + C \rightarrow^{\rho(n)} B + D$

$A + C \rightarrow^{\rho(n)} B + D$

That is:

$A + C \rightarrow^{2\rho(n)} B + D$

Symmetric reactions:

E:

$X = !a;0 \oplus ?a;Y$

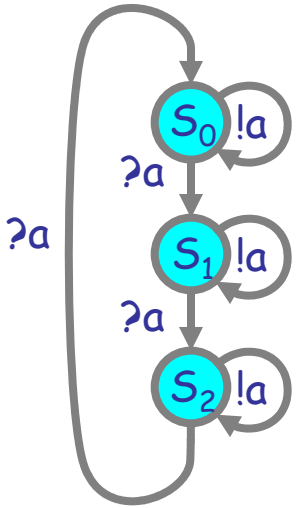
C(E):

$X + X \rightarrow^{2\rho(a)} Y$

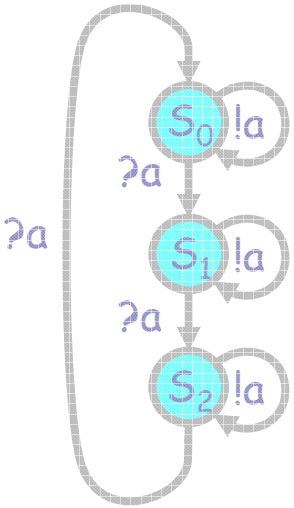
The rate of a was pre-halved and must be restored.

Automata Descriptions are n^2 More Compact

Automaton



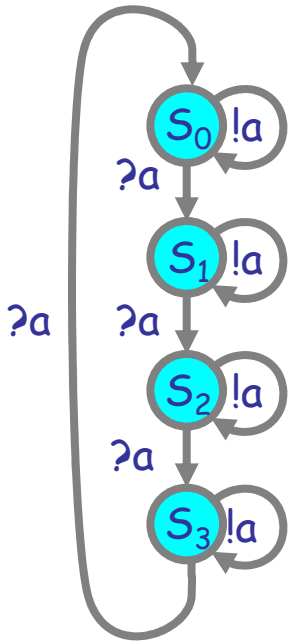
A copy of the automaton to illustrate interactions

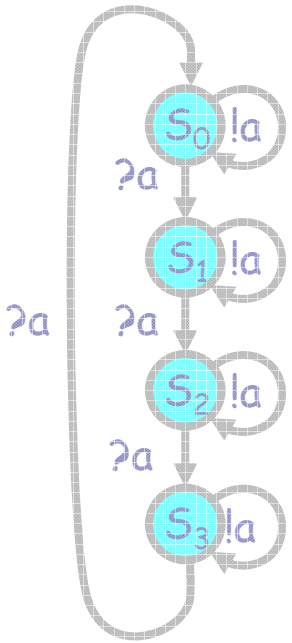


Chemistry

$S_0+S_0 \rightarrow S_0+S_1$
$S_0+S_1 \rightarrow S_0+S_2$
$S_0+S_2 \rightarrow S_0+S_0$
$S_1+S_0 \rightarrow S_1+S_1$
$S_1+S_1 \rightarrow S_1+S_2$
$S_1+S_2 \rightarrow S_1+S_0$
$S_2+S_0 \rightarrow S_2+S_1$
$S_2+S_1 \rightarrow S_2+S_2$
$S_2+S_2 \rightarrow S_2+S_0$

3 states
(2*3 transitions)
=
3^2 reactions





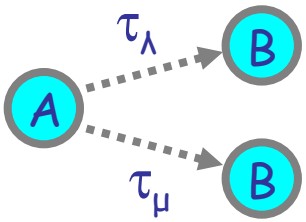
$S_0+S_0 \rightarrow S_0+S_1$
$S_0+S_1 \rightarrow S_0+S_2$
$S_0+S_2 \rightarrow S_0+S_3$
$S_0+S_3 \rightarrow S_0+S_0$
$S_1+S_0 \rightarrow S_1+S_1$
$S_1+S_1 \rightarrow S_1+S_2$
$S_1+S_2 \rightarrow S_1+S_3$
$S_1+S_3 \rightarrow S_1+S_0$
$S_2+S_0 \rightarrow S_2+S_1$
$S_2+S_1 \rightarrow S_2+S_2$
$S_2+S_2 \rightarrow S_2+S_3$
$S_2+S_3 \rightarrow S_2+S_0$
$S_3+S_0 \rightarrow S_3+S_1$
$S_3+S_1 \rightarrow S_3+S_2$
$S_3+S_2 \rightarrow S_3+S_3$
$S_3+S_3 \rightarrow S_3+S_0$

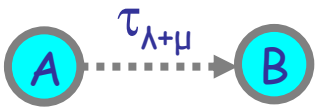
4 states
(2*4 transitions)
=
4^2 reactions

From Processes to ODEs via Chemistry

Choice Law by ODEs

$$\tau_\lambda;B \mid \tau_\mu;B = \tau_{\lambda+\mu};B$$





$A = \tau_\lambda;B \oplus \tau_\mu;B$

$A \to^{\lambda} B$
$A \to^{\mu} B$

$[A]^\bullet = -\lambda[A] - \mu[A]$
$[B]^\bullet = \lambda[A] + \mu[A]$

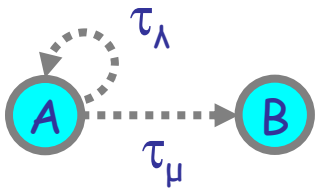
=

$A = \tau_{\lambda+\mu};B$

$A \to^{\lambda+\mu} B$

$[A]^\bullet = -(\lambda+\mu)[A]$
$[B]^\bullet = (\lambda+\mu)[A]$

Idle Loop Law by ODEs

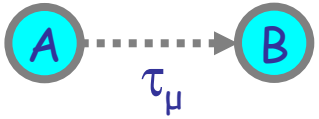


$A = \tau_\lambda;A \oplus \tau_\mu;B$

$A \to^{\lambda} A$
$A \to^{\mu} B$

$[A]^\bullet = -\mu[A]$
$[B]^\bullet = \mu[A]$

=



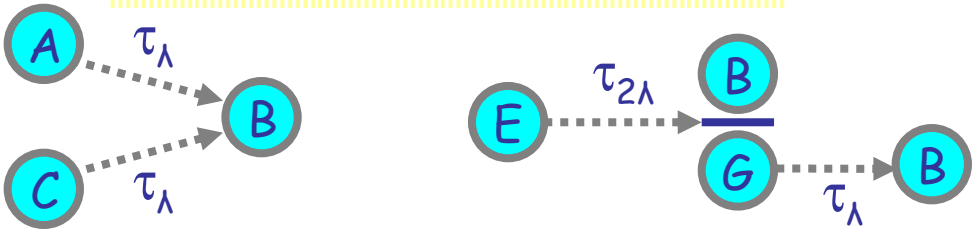
$A = \tau_\mu;B$

$A \to^{\mu} B$

$[A]^\bullet = -\mu[A]$
$[B]^\bullet = \mu[A]$

Equiconfluence Law by ODEs

$\tau_\lambda;B \mid \tau_\lambda;B = \tau_{2\lambda};(B \mid \tau_\lambda;B)$



Want to show that B on both sides has the "same behavior" (equal quantities of B produced at all times)

$F = \tau_\infty;(A \mid C)$
$A = \tau_\lambda;B$
$C = \tau_\lambda;B$

=?

$E = \tau_{2\lambda};(B \mid G)$
$G = \tau_\lambda;B$

$A \to^\lambda B$
$C \to^\lambda B$

=?

$E \to^{2\lambda} B + G$
$G \to^\lambda B$

$[A]^\bullet = -\lambda[A]$
$[C]^\bullet = -\lambda[C]$
$[B]^\bullet = \lambda[A] + \lambda[C]$

$[E]^\bullet = -2\lambda[E]$
$[G]^\bullet = 2\lambda[E] - \lambda[G]$
$[B]^\bullet = 2\lambda[E] + \lambda[G]$

let $A' = C' = E+G/2$

$\lambda[A'] + \lambda[C']$
$= \lambda[E+G/2] + \lambda[E+G/2]$
$= 2\lambda[E] + \lambda[G] = [B]^\bullet$

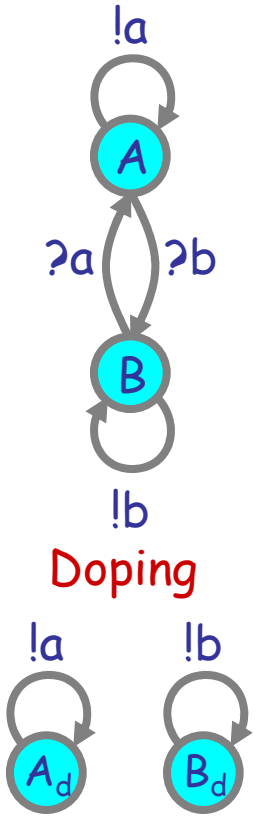
$[A']^\bullet = [E+G/2]^\bullet = [E]^\bullet+[G]^\bullet/2$
$= -2\lambda[E]+(2\lambda[E]-\lambda[G])/2$
$= -\lambda[E]-\lambda[G]/2$
$= -\lambda[E+G/2] = -\lambda[A']$

=

$[A']^\bullet = -\lambda[A']$
$[C']^\bullet = -\lambda[C']$
$[B]^\bullet = \lambda[A'] + \lambda[C']$

Equal ODEs up to a change of variables ($A'=E+G/2$, $C'=E+G/2$), where [B] has the same behavior!

Groupies ODE



$A = !a_{(r)};A \oplus ?b_{(r)};B$ $B = !b_{(r)};B \oplus ?a_{(r)};A$	$A_d = !a_{(r)};A_d$ $B_d = !b_{(r)};B_d$

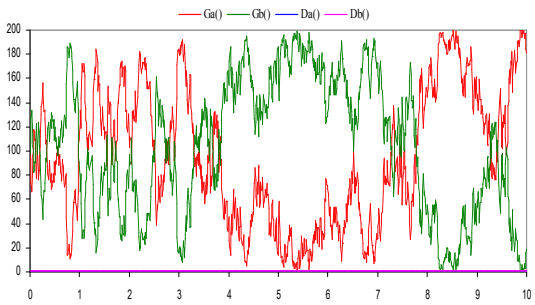
$A+B \rightarrow^r A+A$ $B+A \rightarrow^r B+B$	$A+B_d \rightarrow^r B+B_d$ $B+A_d \rightarrow^r A+A_d$

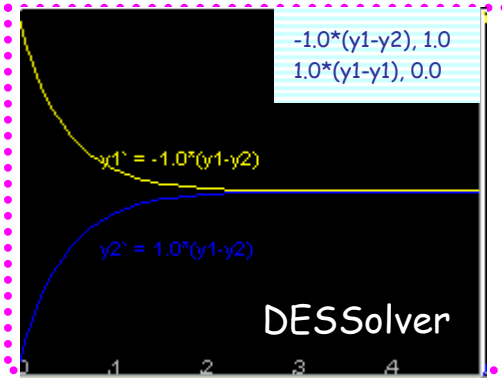
$[A]^\bullet = \cancel{r[A][B]}-\cancel{r[B][A]}-r[A][B_d]+r[B][A_d]$ $[B]^\bullet = \cancel{r[B][A]}-\cancel{r[A][B]}-r[B][A_d]+r[A][B_d]$	$[A_d]^\bullet = 0$ $[B_d]^\bullet = 0$

$[A_d]$,$[B_d]$ are constant; assume them both = k

$$[A]^\bullet = -rk([A]-[B])$$
$$[B]^\bullet = rk([A]-[B])$$

At [B]=0: $[A]^\bullet=-rk[A]$, $[B]^\bullet=rk[A]$
At $[A]\approx[B]$: $[A]^\bullet=[B]^\bullet\approx0$
At [A]=[B]: $[A]^\bullet=[B]^\bullet=0$



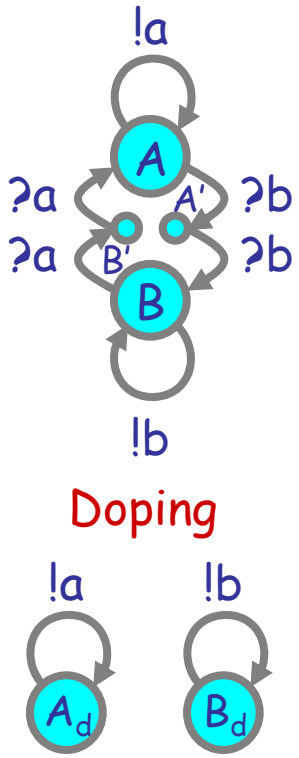


Wrong Answer!

ODE predicts converging stable equilibrium at [A]=[B] instead of the total chaos observed in the stochastic system!

For k=0 (no dope), predicts deadlock $[A]^\bullet=[B]^\bullet=0$ but at any value of [A], which is definitely not true in the stochastic system.

Hysteric Groupies ODE



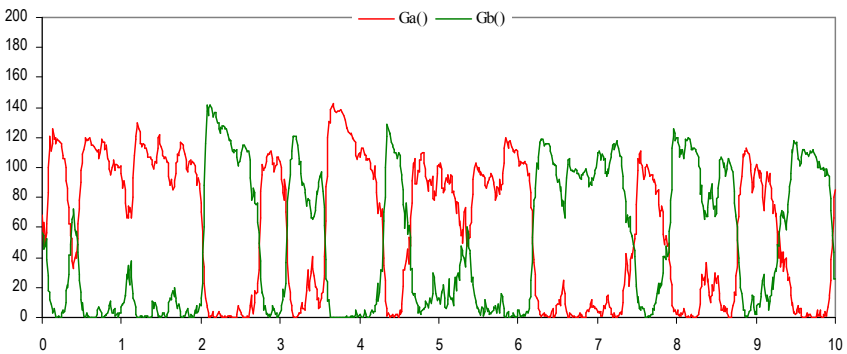
$A = !a_{(r)};A \oplus ?b;A'$ $\quad A' = ?b;B$	$A_d = !a_{(r)};A_d$
$B = !b_{(r)};B \oplus ?a;B'$ $\quad B' = ?a;A$	$B_d = !b_{(r)};B_d$

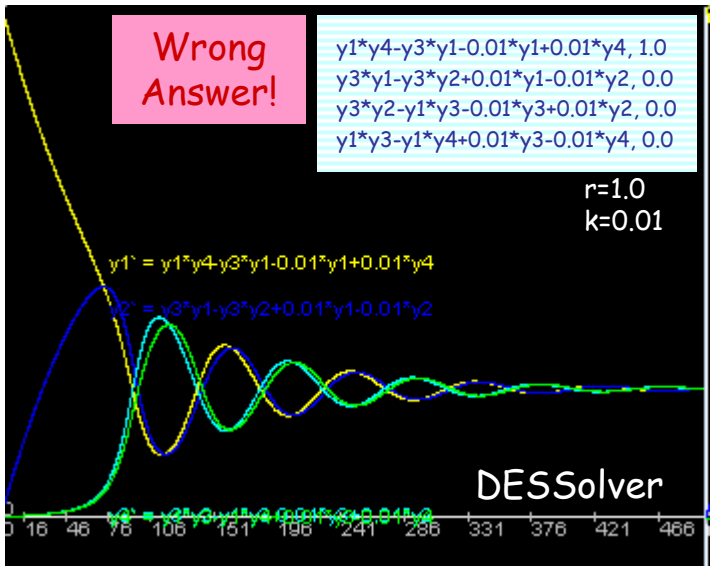
$A+B \to^r A+B' \quad A+B' \to^r A+A$	$A+B_d \to^r A'+B_d \quad A'+B_d \to^r B+B_d$
$B+A \to^r B+A' \quad B+A' \to^r B+B$	$B+A_d \to^r B'+A_d \quad B'+A_d \to^r A+B_d$

$$
\begin{aligned}
[A]^\bullet &= r[A][B']-r[B][A]-r[A][B_d]+r[B'][A_d] \\
[A']^\bullet &= r[B][A]-r[B][A']+r[A][B_d]-r[A'][B_d] \\
[B]^\bullet &= r[B][A']-r[A][B]-r[B][A_d]+r[A'][B_d] \\
[B']^\bullet &= r[A][B]-r[A][B']+r[B][A_d]-r[B'][A_d]
\end{aligned}
\qquad
\begin{aligned}
[A_d]^\bullet &= 0 \\
[B_d]^\bullet &= 0
\end{aligned}
$$

$$
\begin{aligned}
[A]^\bullet &= r[A][B']-r[B][A]-rk[A]+rk[B'] \\
[A']^\bullet &= r[B][A]-r[B][A']+rk[A]-rk[A'] \\
[B]^\bullet &= r[B][A']-r[A][B]-rk[B]+rk[A'] \\
[B']^\bullet &= r[A][B]-r[A][B']+rk[B]-rk[B']
\end{aligned}
$$

$[A_d]$,$[B_d]$ are constant; assume them both = k





Conclusions

Conclusions

- Stochastic Collectives
  - Complex global behavior from simple components
  - Emergence of collective functionality from "non-functional" components
  - (Cf. "swarm intelligence": simple global behavior from complex components)

- Artificial Biochemistry
  - Stochastic collectives with Law of Mass Interaction kinetics
  - Connections to classical Markov theory, chemical Master Equation, and Rate Equation

- The agent/automaton/process point of view
  - *Individuals* that transition between states (vs. *transmutation* between unrelated chemical species)
  - More appropriate for Systems Biology
  - Stochastic π-calculus (SPiM) for investigating stochastic collectives
    - Restriction+Communication ⇒ Polymerization: FSA that "stick together"

- Properties of collective behavior
  - ??