Artificial Biochemistry
Auxiliary Material

Luca Cardelli

Microsoft Research
Cambridge UK

2008-02

http://LucaCardelli.name

Section 2

Fig 1. Interacting Automata

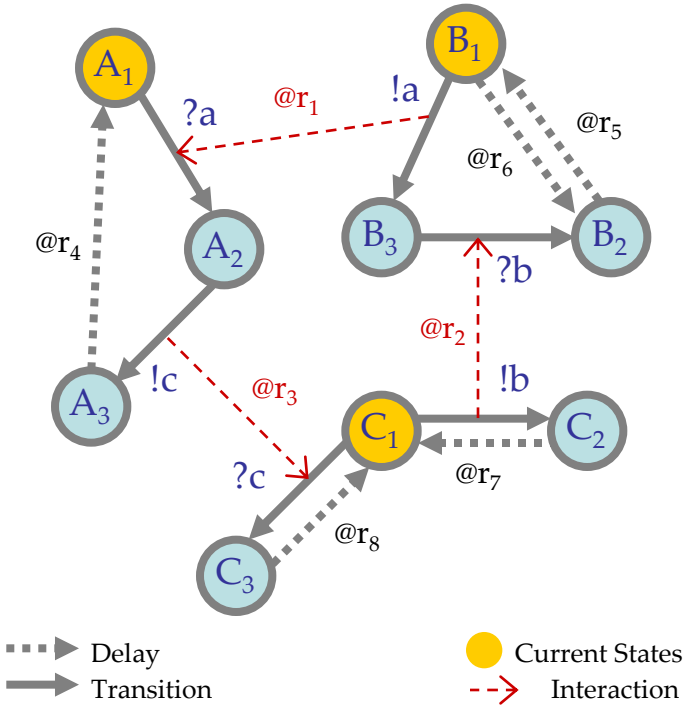


Fig. 1. Interacting automata

Fig 2. Automata Reactions

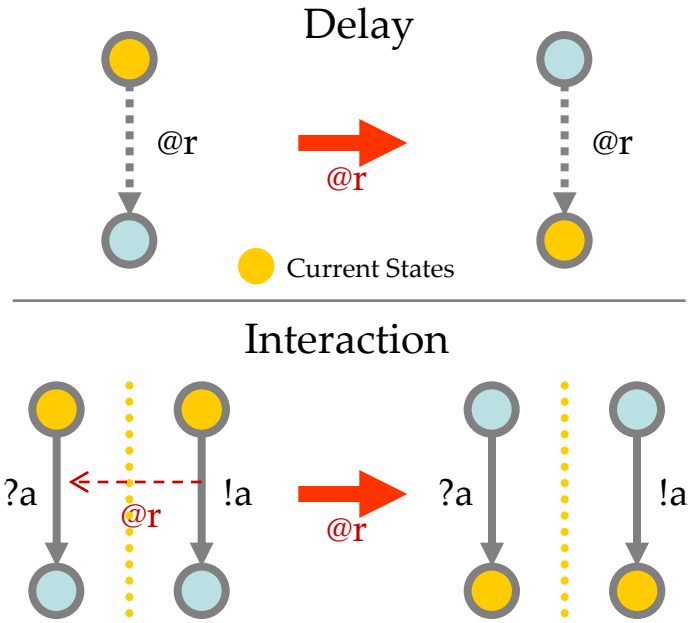


Fig. 2. Automata reactions

Fig 3. Celebrity Automata

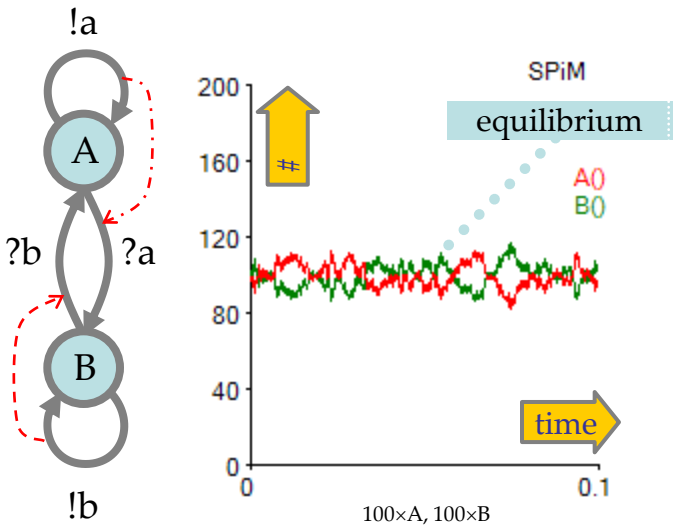


Fig. 3. Celebrity automata

```
directive sample 0.1
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())
```

Fig 4. Possible Interactions

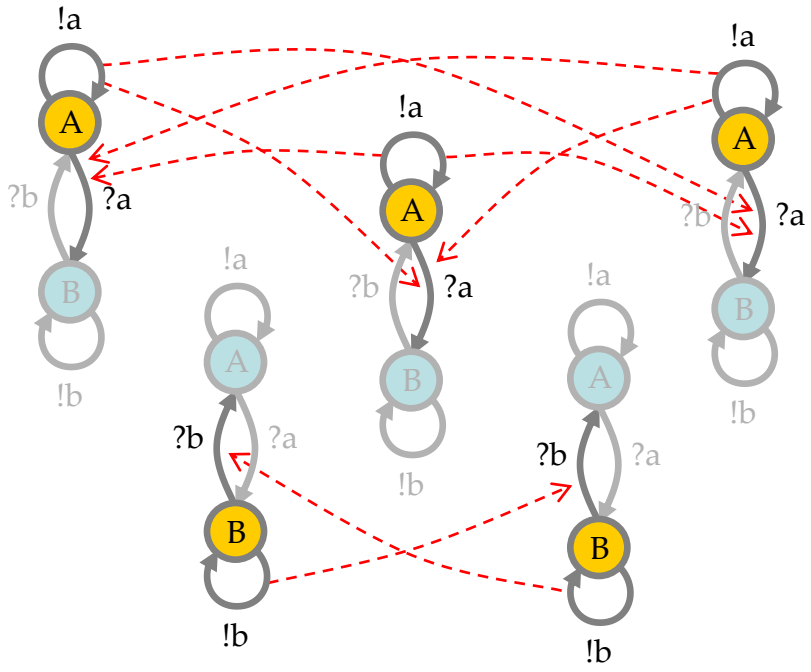
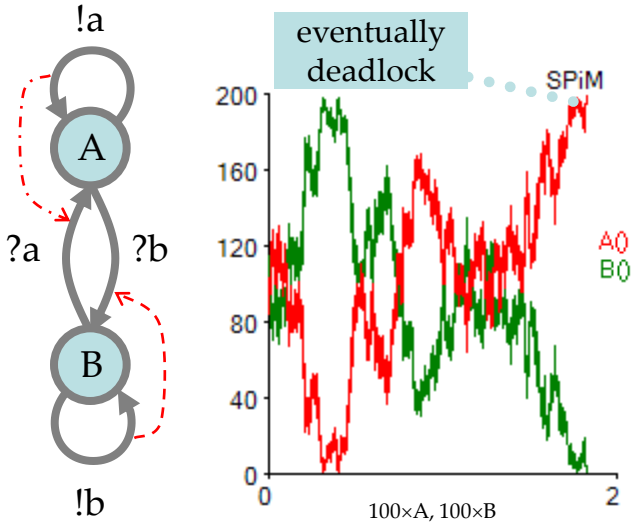


Fig. 4. Possible interactions

Fig 5. Groupie Automata




```
directive sample 2.0
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())
```

Fig. 5. Groupie automata

Fig 6. Both Together

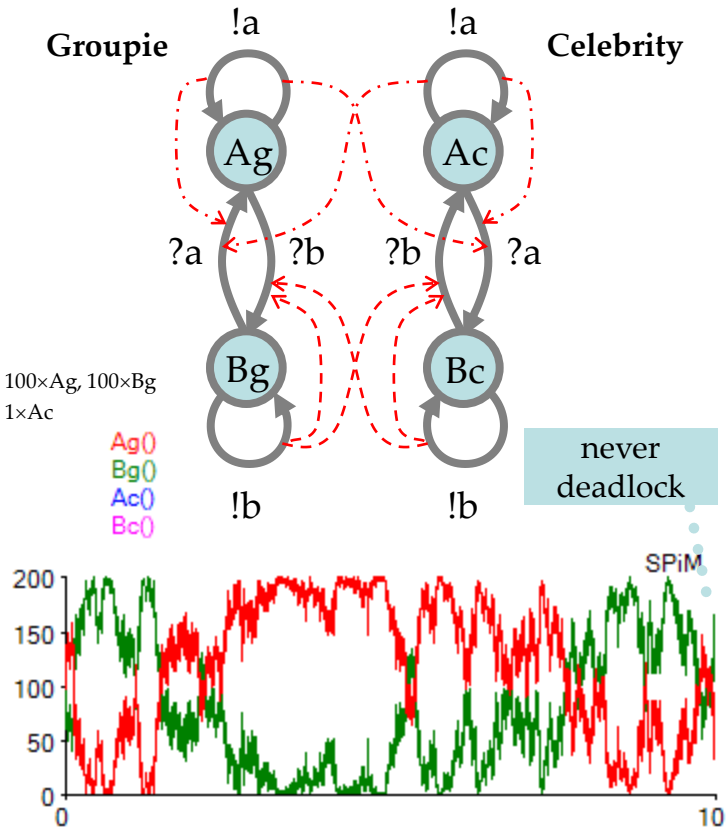
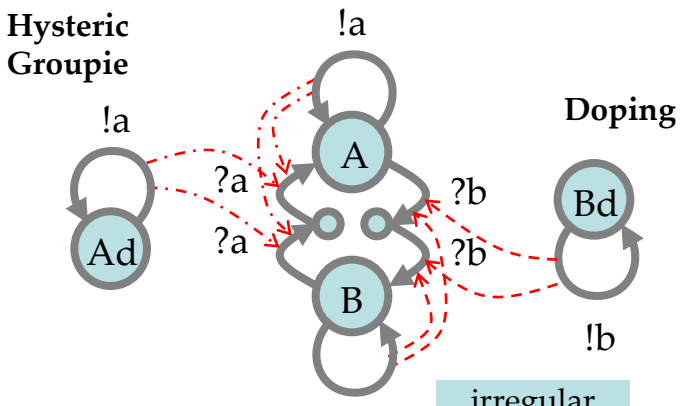


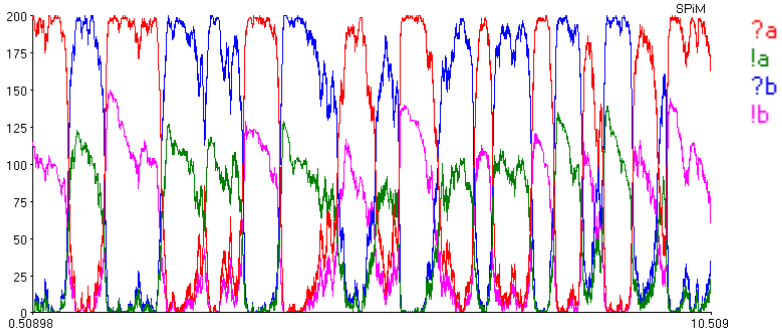
Fig. 6. Both together

```
directive sample 10.0 1000
directive plot Ag(); Bg(); Ac(); Bc()

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

let Ac() = do !a; Ac() or ?a; Bc()
and Bc() = do !b; Bc() or ?b; Ac()

let Ag() = do !a; Ag() or ?b; Bg()
and Bg() = do !b; Bg() or ?a; Ag()

run 1 of Ac()
run 100 of (Ag() | Bg())
```

Hysteric Groupies (1 Mid-State)

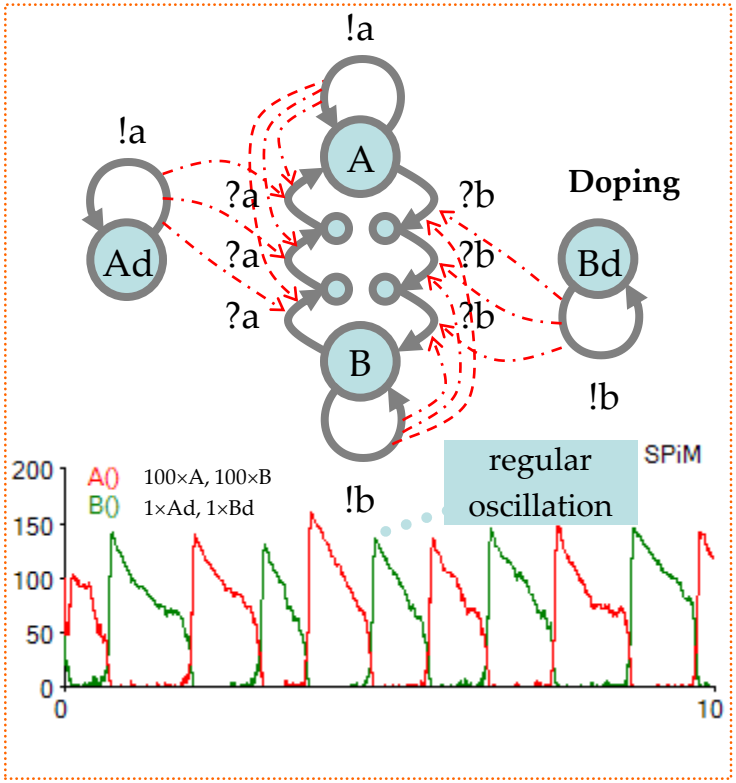



```
directive sample 10.0  1000
directive plot A(); B()

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

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

let Ad() = !a; Ad()
and Bd() = !b; Bd()

run 100 of (A() | B())
run 1 of (Ad() | Bd())
```

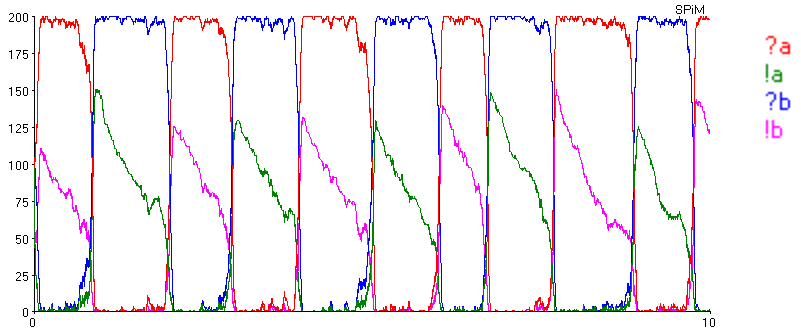


Plotting also the mid-states.

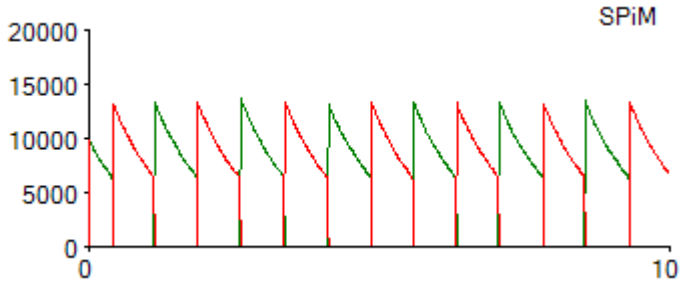
Hysteric Groupies (2 Mid-States)




```
directive sample 10.0 1000
directive plot A(); B()

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 Ad() = !a; Ad()
and Bd() = !b; Bd()

run 100 of (A() | B())
run 1 of (Ad() | Bd())
```



Plotting also the mid-states.



Higher precision at higher numbers.

Fig 7. Hysteric Groupies

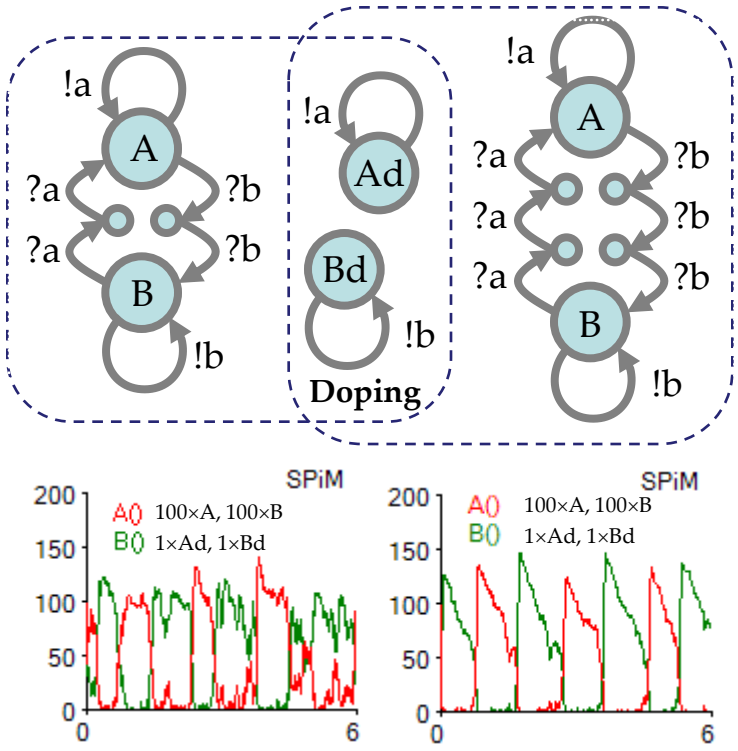


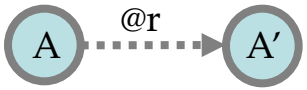
Fig. 7. Hysteric groupies

```
directive sample 6.0  1000
directive plot A(); B()

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

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

let Ad() = !a; Ad()
and Bd() = !b; Bd()

run 100 of (A() | B())
run 1 of (Ad() | Bd())
```

```
directive sample 6.0 1000
directive plot A(); B()

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 Ad() = !a; Ad()
and Bd() = !b; Bd()

run 100 of (A() | B())
run 1 of (Ad() | Bd())
```

Section 3.2

Fig 8. First Order Reactions



$A \rightarrow^{r} A'$

Fig. 8. First order reactions

Fig 9. Sequence of delays

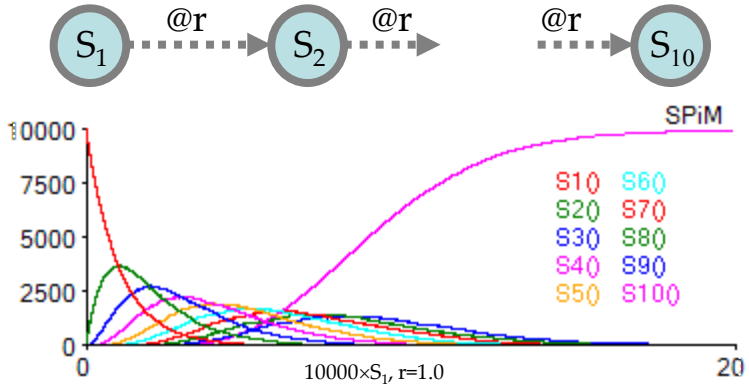
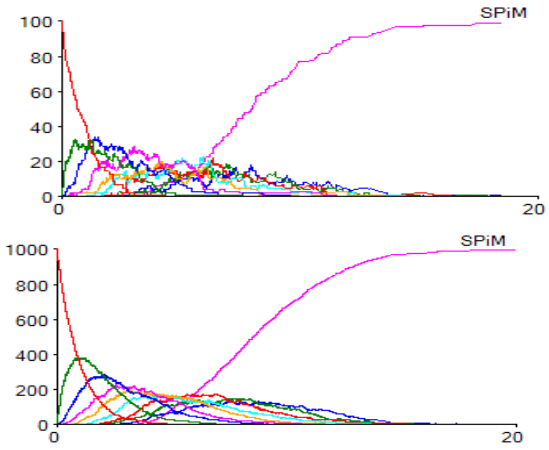


Fig. 9. Sequence of delays

```
directive sample 20.0
directive plot S1(); S2(); S3(); S4(); S5(); S6(); S7(); S8(); S9(); S10()

let S1() = delay@1.0; S2()
and S2() = delay@1.0;S3()
and S3() = delay@1.0; S4()
and S4() = delay@1.0; S5()
and S5() = delay@1.0; S6()
and S6() = delay@1.0; S7()
and S7() = delay@1.0; S8()
and S8() = delay@1.0; S9()
and S9() = delay@1.0; S10()
and S10() = ()

run 10000 of S1()
```



Time scale is independent of initial quantities.

Section 3.3

Fig 10. Second Order Reactions

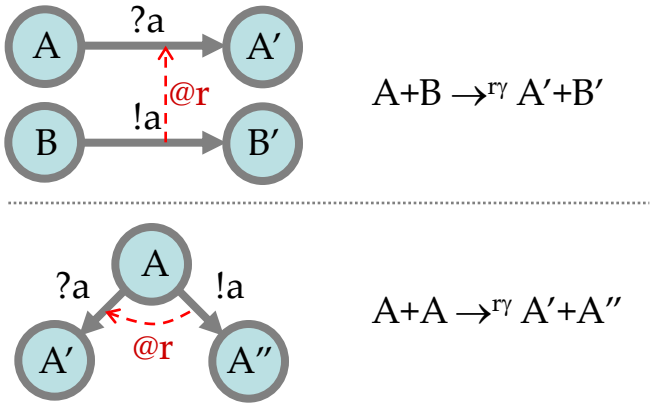


Fig. 10. Second order reactions

Fig 11. All 3 Reactions in 1 Automaton

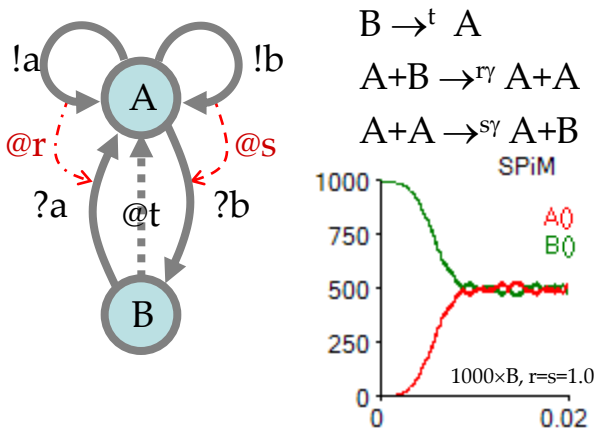


Fig. 11. All 3 reactions

```
directive sample 0.02
directive plot A(); B()

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

let A() = do !a; A() or !b; A() or ?b; B()
and B() = do delay@1.0; A() or ?a; A()

run 1000 of  B()
```

Fig 12. Different Reactions, Same Behavior

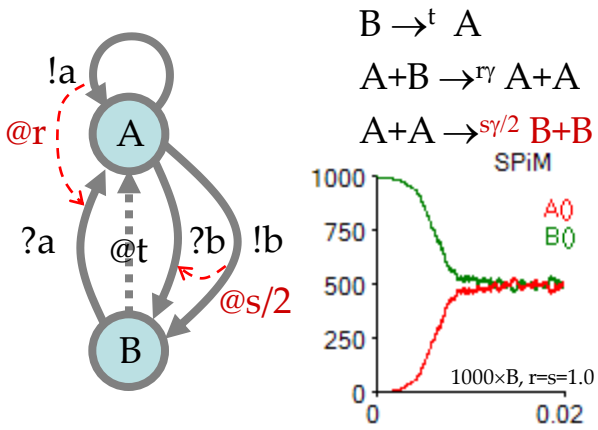


Fig. 12. Same behavior

```
directive sample 0.02
directive plot A(); B()

new a@1.0:chan()
new b@0.5:chan()

let A() = do !a; A() or !b; B() or ?b; B()
and B() = do delay@1.0; A() or ?a; A()

run 1000 of  B()
```

Fig 13. Sequence of Interactions

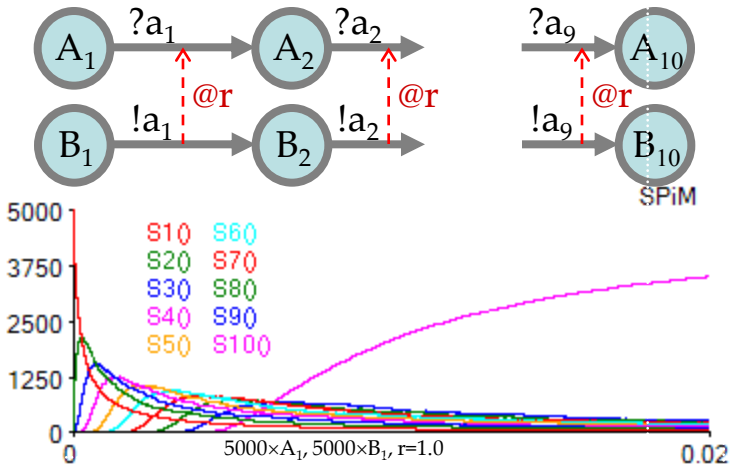
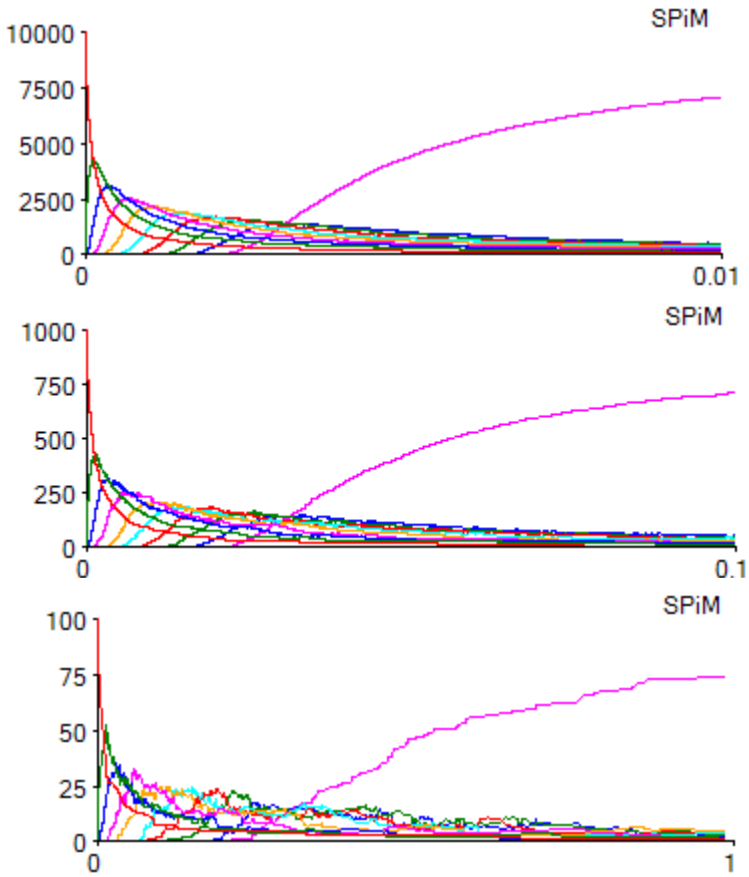


Fig. 13. Sequence of interactions

```
directive sample 0.02
directive plot A1(); A2(); A3(); A4(); A5(); A6(); A7(); A8(); A9(); A10()

new a1@1.0:chan new a2@1.0:chan new a3@1.0:chan new a4@1.0:chan new a5@1.0:chan new a6@1.0:chan
new a7@1.0:chan new a8@1.0:chan new a9@1.0:chan

let A1() = ?a1; A2()
and B1() = !a1; B2()
and A2() = ?a2; A3()
and B2() = !a2; B3()
and A3() = ?a3; A4()
and B3() = !a3; B4()
and A4() = ?a4; A5()
and B4() = !a4; B5()
and A5() = ?a5; A6()
and B5() = !a5; B6()
and A6() = ?a6; A7()
and B6() = !a6; B7()
and A7() = ?a7; A8()
and B7() = !a7; B8()
and A8() = ?a8; A9()
and B8() = !a8; B9()
and A9() = ?a9; A10()
and B9() = !a9; B10()
and A10() = ()
and B10() = ()

run 5000 of (A1() | B1())
```

Time scale is *decreases* linearly depending on quantity



Section 3.4

Fig 14. Zero Order Reactions

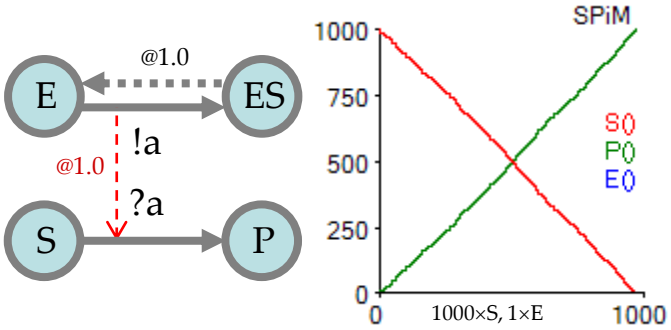


Fig. 14. Zero order reactions

```
directive sample 1000.0
directive plot S(); P(); E()

new a@1.0:chan()

let E() = !a; delay@1.0; E()
and S() = ?a; P()
and P() = ()

run (1 of  E() | 1000 of S())
```

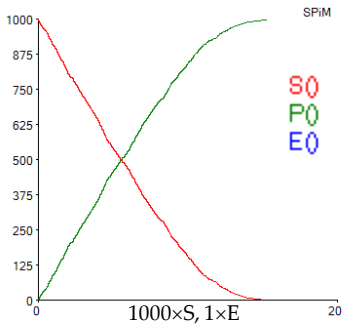
$ES \rightarrow^{s} E$

$E+S \rightarrow^{r} ES+P$

$K_m = 1$

$V_{max} = [E_0]$

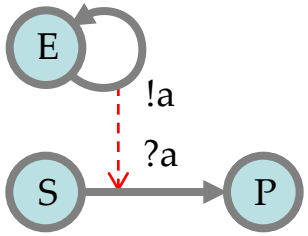
Zero Order Regime

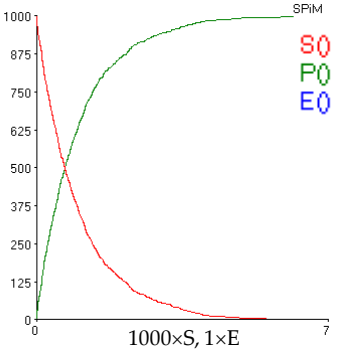



```
directive sample 0.2
directive plot S(); P(); E()

new a@1.0:chan()

let E() = !a; delay@100.0; E()
and S() = ?a; P()
and P() = ()

run (1 of  E() | 1000 of S())
```

$K_m = 100$

Intermediate Regime



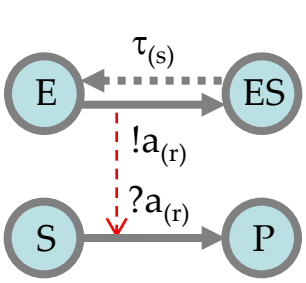


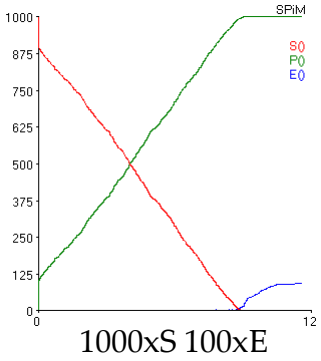

```
directive sample 1000.0
directive plot S(); P(); E()

new a@1.0:chan()

let E() = !a; E()
and S() = ?a; P()
and P() = ()

run (1 of  E() | 1000 of S())
```

$E+S \rightarrow^{r} E+P$

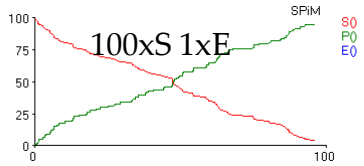
Second Order Regime

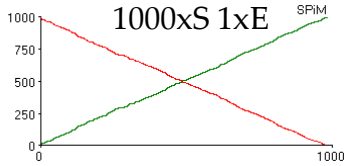
Zero Order Reaction Scaling

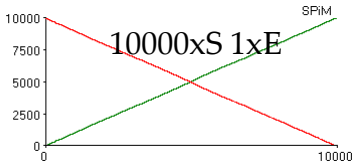




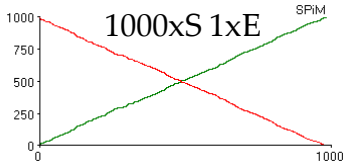

```
directive sample 100.0
directive plot S(); P(); E()

new a@1.0:chan()

let E() = !a; delay@1.0; E()
and S() = ?a; P()
and P() = ()

run (100 of  E() | 1000 of S())
```

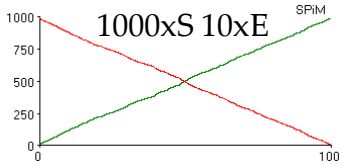


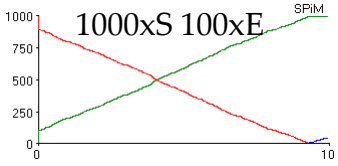




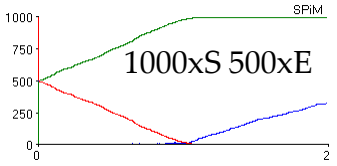
Time scale *increases* linearly depending on quantity of S

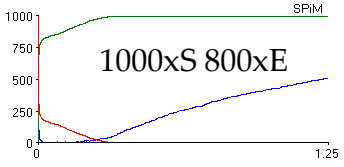


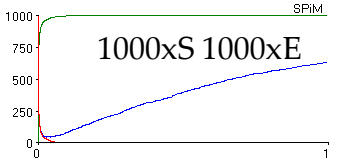




Time scale *decreases* linearly depending on quantity of E







Up to the point where E>S

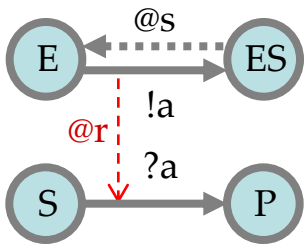
Pseudo Michaelis-Menten Derivation

$[E]^\bullet = s[ES] - r[E][S]$

$[ES]^\bullet = r[E][S] - s[ES]$

$[S]^\bullet = - r[E][S]$

$[P]^\bullet = r[E][S]$



$ES \rightarrow^{s} E$

$E+S \rightarrow^{r} ES+P$

Assume $[ES]^\bullet = 0$

$[ES] = r[E][S]/s$

define $K_m = s/r$

$[ES] = [E][S]/K_m$ (1)

$[P]^\bullet = r[E][S] = s[ES]$ (2)

Total enzyme is

$[E_0] = [E]+[ES]$ hence

$[E] = [E_0] - [ES]$ (3)

Substituting (3) into (1)

$[ES] = ([E_0] - [ES])[S]/K_m$ i.e.

$[E_0] = K_m[ES]/[S] + [ES]$

$[E_0] = [ES](1+K_m/[S])$

$[ES]= [E_0]/(1+K_m/[S])$

mult. num&denum by [S]:

$[ES] = [E_0]([S]/(K_m+[S]))$ (4)

Substituting (4) into (2)

$[P]^\bullet = s[E_0]([S]/(K_m+[S]))$

define $V_{max} = s[E_0]$

$[P]^\bullet = V_{max}[S]/(K_m+[S])$ (5)

Real Michaelis-Menten:

$E+S \rightarrow^{r} ES$

$ES \rightarrow^{s1} E+S$

$ES \rightarrow^{s2} E+P$

$K_m = (s1+s2)/r$

$V_{max} = s2[E_0]$

so if $s1 \ll s2$

we have $s \approx s2$

See http://en.wikipedia.org/wiki/Michaelis-Menten_kinetic

Section 3.5

Fig 15. Subtraction

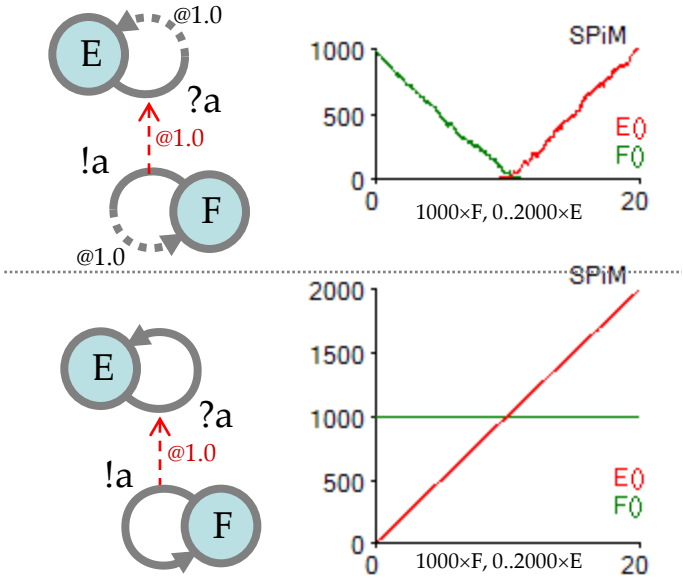


Fig. 15. Subtraction (top)

```
directive sample 20.0 1000
directive plot E(); F()

new a@1.0:chan()

let E() = ?a; delay@1.0; E()
and F() = !a; delay@1.0; F()

let raising(p:proc(), t:float) = (* Producing one p() every t sec *)
  (val dt= 100.0 run step(p, t, dt, dt))        (* with precision dt *)
and step(p:proc(), t:float, n:float, dt:float) =
  if n<=0.0 then (p()|step(p,t,dt,dt)) else delay@dt/t; step(p,t,n-1.0,dt)
run 1000 of F()
run raising(E,0.01)
```

```
directive sample 20.0 1000
directive plot E(); F()

new a@1.0:chan()

let E() = ?a; E()
and F() = !a; F()

let raising(p:proc(), t:float) = (* Producing one p() every t sec *)
  (val dt= 100.0 run step(p, t, dt, dt))        (* with precision dt *)
and step(p:proc(), t:float, n:float, dt:float) =
  if n<=0.0 then (p()|step(p,t,dt,dt)) else delay@dt/t; step(p,t,n-1.0,dt)
run 1000 of F()
run raising(E,0.01)
```

Fig 16. Ultrasensitivity

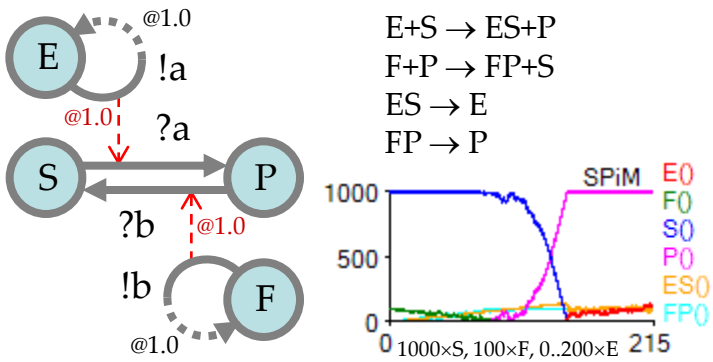
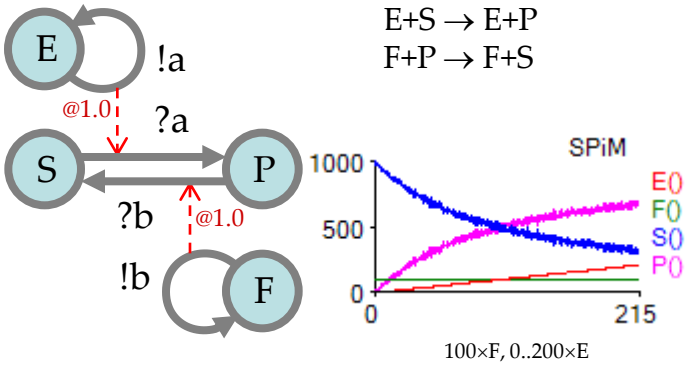


Fig. 16. Ultrasensitivity

```
directive sample 215.0
directive plot E(); F(); S(); P(); ES(); FP()

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

let S() = ?a; P()
and P() = ?b; S()

let E() = !a; ES()
and ES() = delay@1.0; E()
and F() = !b; FP()
and FP() = delay@1.0; F()

let raising(p:proc(), t:float) = (* Producing one p() every t sec *)
  (val dt= 100.0 run step(p, t, dt, dt))        (* with precision dt *)
and step(p:proc(), t:float, n:float, dt:float) =
  if n<=0.0 then (p()|step(p,t,dt,dt)) else delay@dt/t; step(p,t,n-1.0,dt)

run 1000 of S()
run 100 of F()
run raising(E,1.0)
```

Compare with Second-Order Regime




```
directive sample 215.0 1000
directive plot E(); F(); S(); P()

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

let S() = ?a; P()
and P() = ?b; S()

let E() = !a; E()
and F() = !b; F()

let raising(p:proc(), t:float) = (* Producing one p() every t sec *)
  (val dt= 100.0 run step(p, t, dt, dt))        (* with precision dt *)
and step(p:proc(), t:float, n:float, dt:float) =
  if n<=0.0 then (p()|step(p,t,dt,dt)) else delay@dt/t; step(p,t,n-1.0,dt)

run 1000 of S()
run 100 of F()
run raising(E,1.0)
```

Section 3.6

Fig 17. Positive Feedback Transition

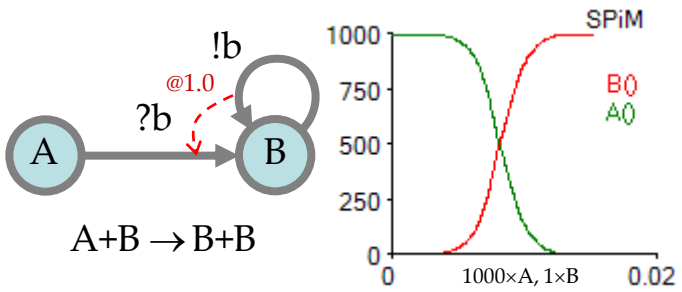


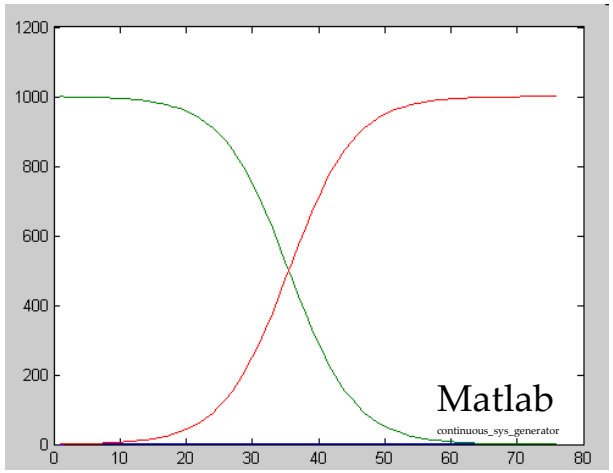
Fig. 17. Positive feedback transition

```
directive sample 0.02 1000
directive plot B(); A()

val s=1.0

new b@s:chan
let A() = ?b; B()
and B() = !b;B()

run (1000 of A() | 1 of B())
```

N.B.: needs at least 1 B to "get started".

$$A+B \rightarrow^{s} B+B$$

$$[A]^{\bullet} = -s[A][B]$$
$$[B]^{\bullet} = s[A][B]$$




```
interval/step [0:0.001:0.0]
(A)     dx1/dt = - x1*x2 1000.0
(B)     dx2/dt = x1*x2   1.0
```

Fig 18. Bell Shape

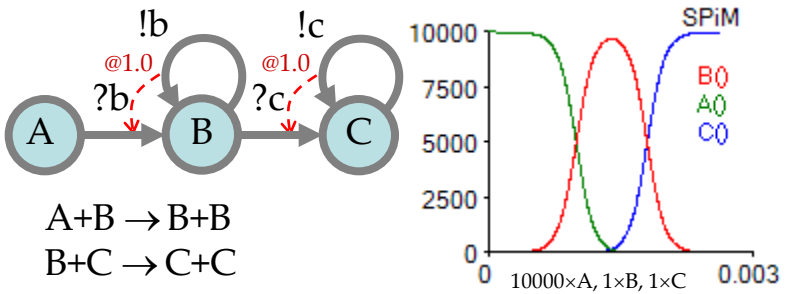
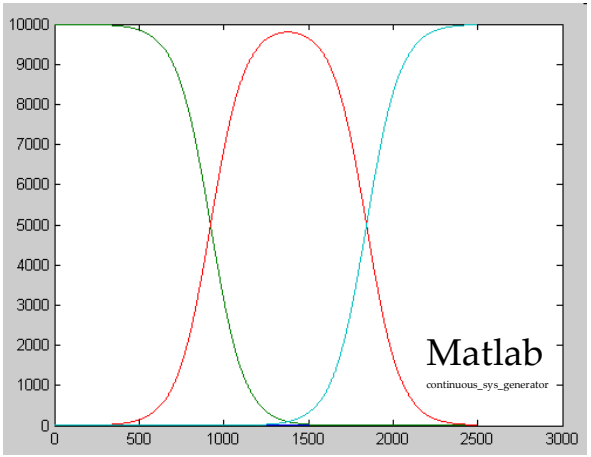


Fig. 18. Bell shape

```
directive sample 0.003 1000
directive plot B(); A(); C()

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

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

run ((10000 of A()) | B() | C())
```

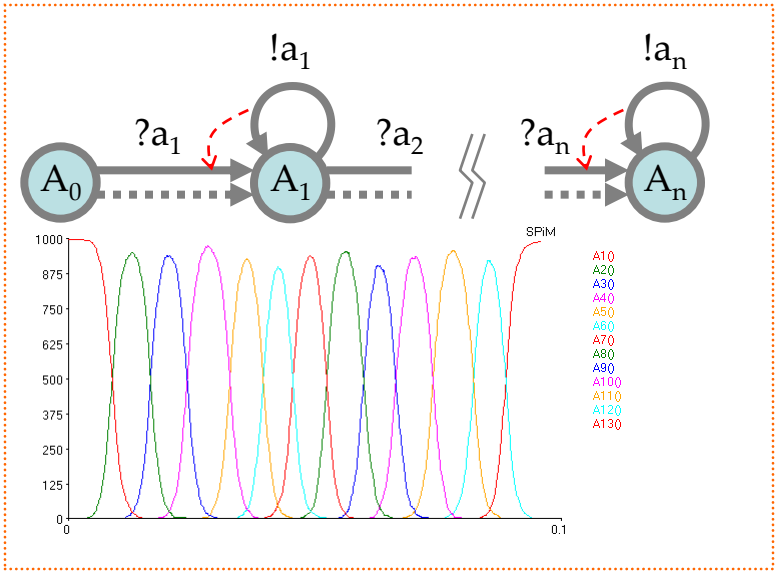
$A+B \rightarrow^{1} B+B$
$B+C \rightarrow^{1} C+C$

$[A]^{\bullet} = -[A][B]$
$[B]^{\bullet} = [A][B]-[B][C]$
$[C]^{\bullet} = [B][C]$




```
interval/step [0:0.000001:0.0025]
(A)   dx1/dt = -x1*x2          10000.0
(B)   dx2/dt = x1*x2 – x2*x3   1.0
(C)   dx3/dt = x2*x3           1.0
```

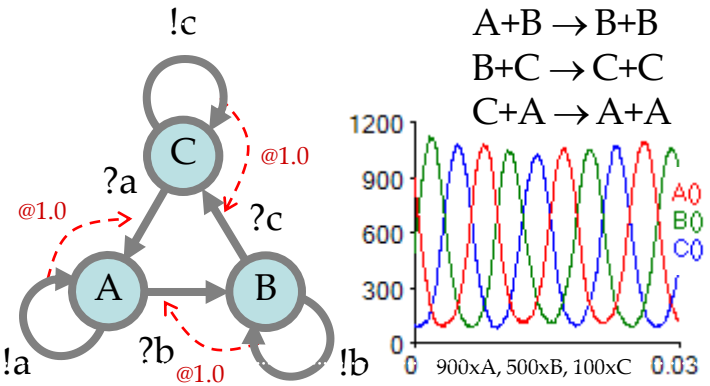
Soliton-like Propagation




```
directive sample 0.1 1000
directive plot A1(); A2(); A3(); A4(); A5(); A6(); A7(); A8();
A9(); A10(); A11(); A12(); A13()

val r=1.0 val s=1.0

new a2@s:chan new a3@s:chan new a4@s:chan
new a5@s:chan new a6@s:chan new a7@s:chan
new a8@s:chan new a9@s:chan new a10@s:chan
new a11@s:chan new a12@s:chan new a13@s:chan
let A1() = do delay@r;A2() or ?a2; A2()
and A2() = do !a2;A2() or delay@r;A3() or ?a3; A3()
and A3() = do !a3;A3() or delay@r;A4() or ?a4; A4()
and A4() = do !a4;A4() or delay@r;A5() or ?a5; A5()
and A5() = do !a5;A5() or delay@r;A6() or ?a6; A6()
and A6() = do !a6;A6() or delay@r;A7() or ?a7; A7()
and A7() = do !a7;A7() or delay@r;A8() or ?a8; A8()
and A8() = do !a8;A8() or delay@r;A9() or ?a9; A9()
and A9() = do !a9;A9() or delay@r;A10() or ?a10; A10()
and A10() = do !a10;A10() or delay@r;A11() or ?a11; A11()
and A11() = do !a11;A11() or delay@r;A12() or ?a12; A12()
and A12() = do !a12;A12() or delay@r;A13() or ?a13; A13()
and A13() = !a13;A13()

run 1000 of A1()
```

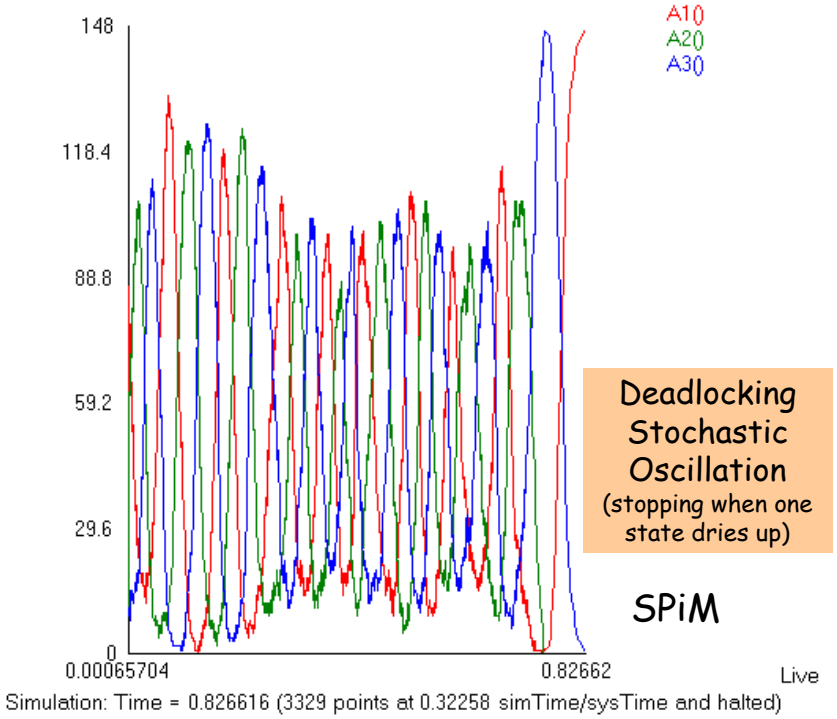
Initial pulse propagates without loosing shape.

Fig 19. Oscillator




```
directive sample 0.03 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 ?b; B()
and B() = do !b;B() or ?c; C()
and C() = do !c;C() or ?a; A()

run (900 of A() | 500 of B() | 100 of C())
```

Fig. 19. Oscillator



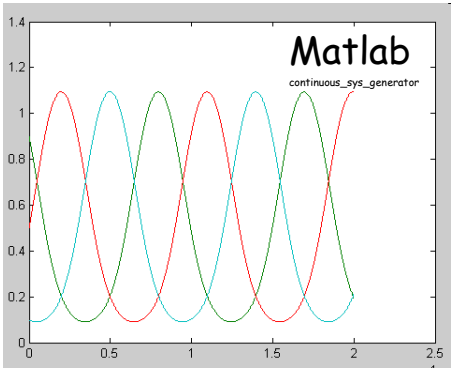
Deadlocking Stochastic Oscillation
(stopping when one state dries up)

SPiM

Numerical Instabilities in Matlab Solvers

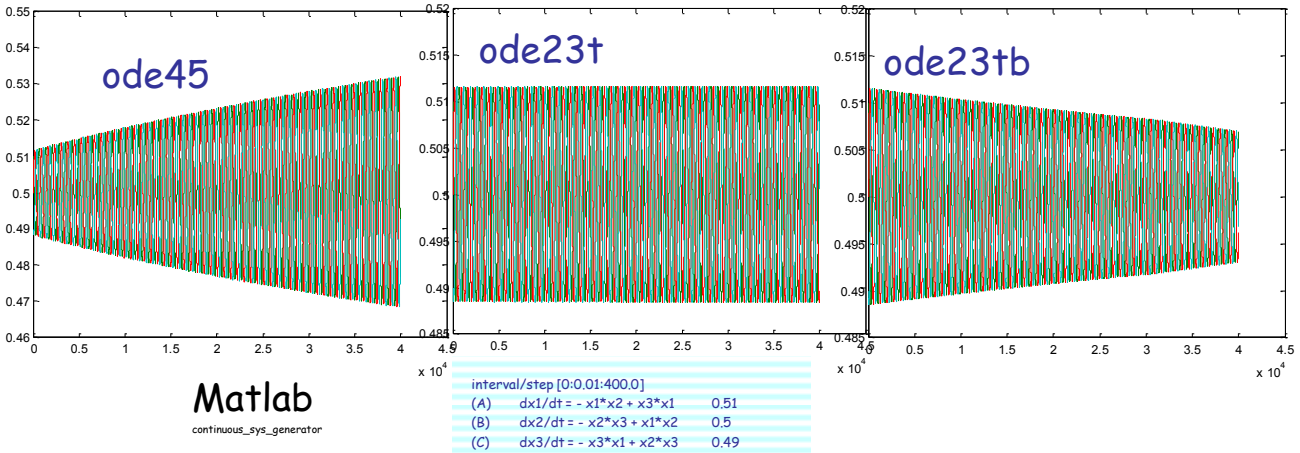
$A+B \rightarrow^{s} B+B$
$B+C \rightarrow^{s} C+C$
$C+A \rightarrow^{s} A+A$

$[A]^{\bullet} = -s[A][B]+s[C][A]$
$[B]^{\bullet} = -s[B][C]+s[A][B]$
$[C]^{\bullet} = -s[C][A]+s[B][C]$



interval/step [0:0.001:20.0]
(A) dx1/dt = - x1*x2 + x3*x1 0.9
(B) dx2/dt = - x2*x3 + x1*x2 0.5
(C) dx3/dt = - x3*x1 + x2*x3 0.1

N.B.: all rates 1.0



interval/step [0:0.01:400.0]
(A) dx1/dt = - x1*x2 + x3*x1 0.51
(B) dx2/dt = - x2*x3 + x1*x2 0.5
(C) dx3/dt = - x3*x1 + x2*x3 0.49

Fig 20. Positive two-stage Feedback

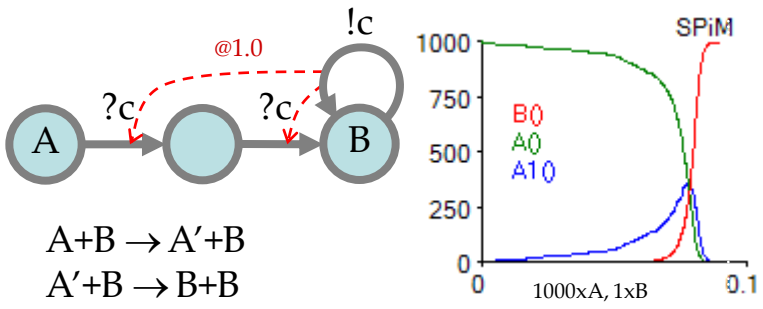
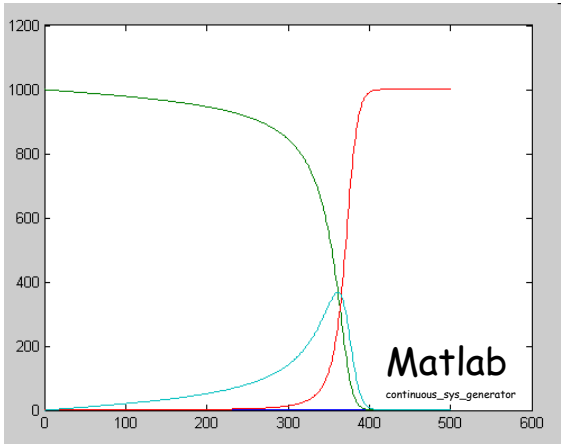


Fig. 20. Positive two-stage feedback

```
directive sample 0.1 1000
directive plot B(); A(); A1()

new c@1.0:chan
let A() = ?c; A1()
and A1() = ?c; B()
and B() = !c;B()

run (1000 of A() | 1 of B())
```

$A+B \rightarrow^{s} A'+B$
$A'+B \rightarrow^{s} B+B$

$[A]^{\bullet} = -s[A][B]$
$[B]^{\bullet} = s[A'][B]$
$[A']^{\bullet} = s[A][B]-s[A'][B]$




```
interval/step [0:0.0002:0.1]
(A)  dx1/dt = - x1*x2          1000.0
(B)  dx2/dt = x3*x2            1.0
(A') dx3/dt = x1*x2 - x3*x2    0.0
```

Fig 21. Square Shape

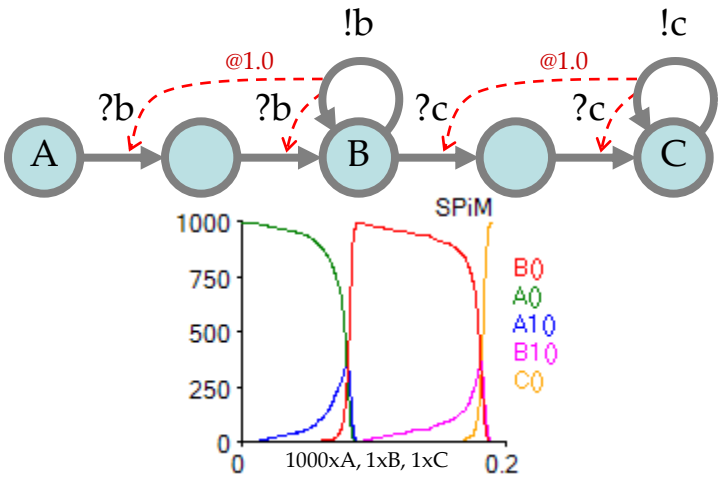
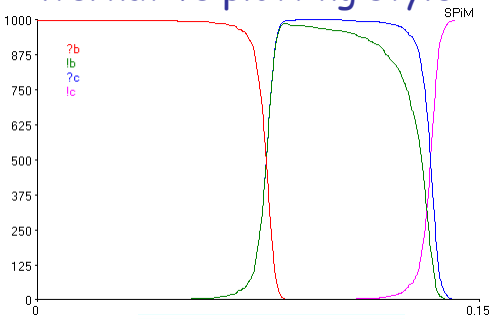


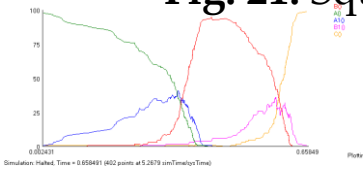
Fig. 21. Square shape

```
directive sample 0.2 1000
directive plot B(); A(); A1(); B1(); C()

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

let A() = ?b; A1()
and A1() = ?b; B()
and B() = do !b;B() or ?c; B1()
and B1() = ?c; C()
and C() = !c;C()

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

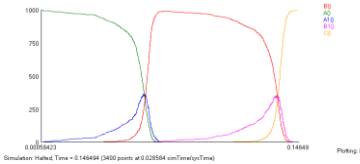
Alternative plotting style




```
directive sample 015 1000
(* directive plot B(); A(); A1(); B1(); C() *)

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

let A() = ?b; A1()
and A1() = ?b; B()
and B() = do !b;B() or ?c; B1()
and B1() = ?c; C()
and C() = !c;C()

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

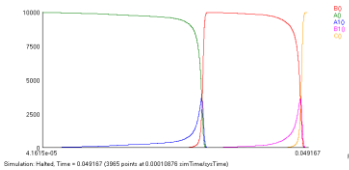


100xA



1000xA

```
directive sample 0.06 1000
directive plot B(); A(); A1(); B1(); C()

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

let A() = ?b; A1()
and A1() = ?b; B()
and B() = do !b;B() or ?c; B1()
and B1() = ?c; C()
and C() = !c;C()

run ((10000 of A()) | B() | C())
```



10000xA

N.B.: starting with 10000xA, 1xB, 1xC: insignificant chance that B will react with C first and the whole thing will starve.

Fig 22. Hysteric 3-Way Groupies

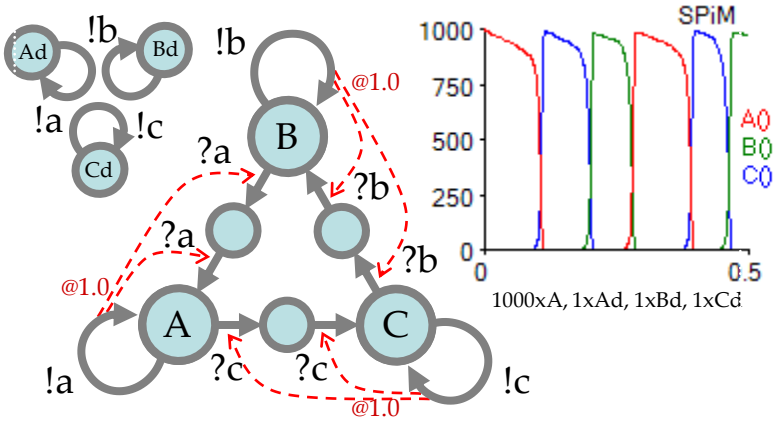


Fig. 22. Hysteric 3-way groupies

```
directive sample 0.5 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 Ad() = !a; Ad()
and Bd() = !b; Bd()
and Cd() = !c; Cd()

run 1000 of A()
run 1 of (Ad() | Bd() | Cd())
```

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

Section 3.7

Fig 23. Second-Order Cascade

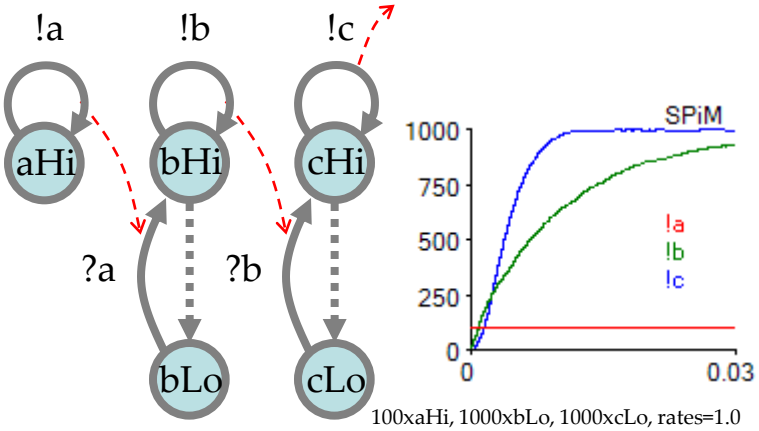


Fig. 23. Second-order cascade

```
directive sample 0.03
directive plot !a; !b; !c

new a@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)

run 1000 of (Amp_lo(a,b) | Amp_lo(b,c))

let A() = !a; A()
run 100 of A()
```

Fig 24. Zero-order Cascade

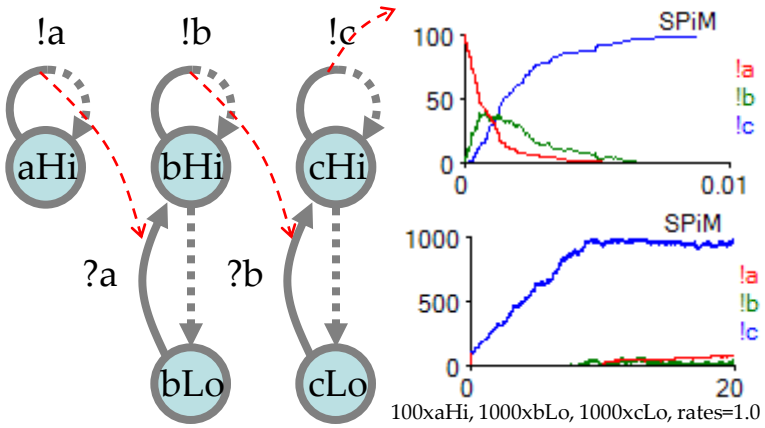


Fig. 24. Zero-order cascade

```
directive sample 0.01
directive plot !a; !b; !c

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

let Amp_hi(a:chan, b:chan) =
  do !b; delay@1.0; 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 1000 of (Amp_lo(a,b) |
Amp_lo(b,c))

let A() = !a; delay@1.0; A()
run 100 of A()
```

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

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

let Amp_hi(a:chan, b:chan) =
  do !b; delay@1.0; 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 1000 of (Amp_lo(a,b) |
Amp_lo(b,c))

let A() = !a; delay@1.0; A()
run 100 of A()
```

Fig 25. Zero-order Transduction

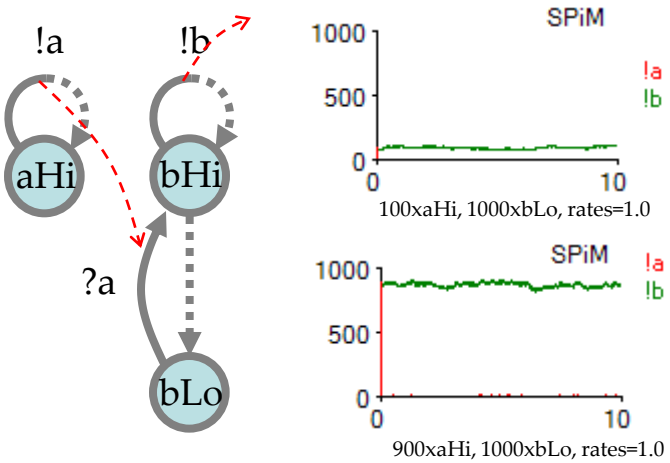


Fig. 25. Zero-order cascade - 1 stage

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

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

let Amp_hi(a:chan, b:chan) =
  do !b; delay@1.0; 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 1000 of Amp_lo(a,b)

let A() = !a; delay@1.0; A()
run 500 of A()
```

Fig 26. Second-Order Double Cascade

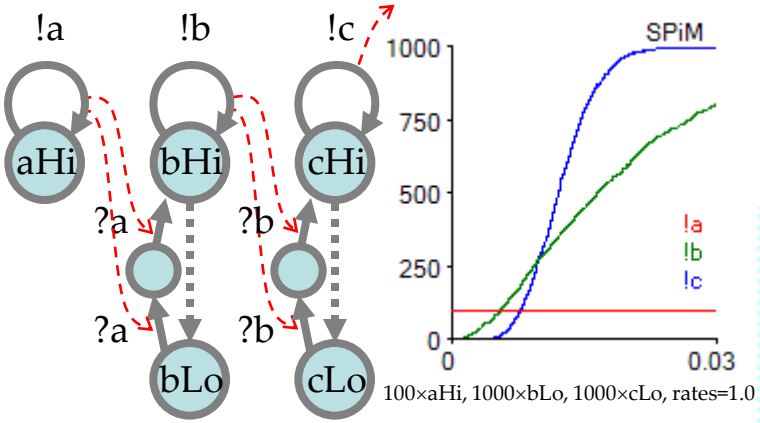
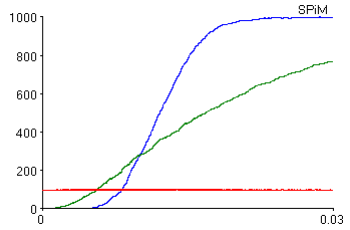


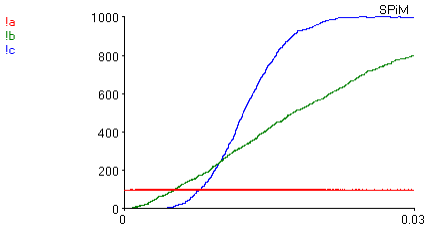
Fig. 26. Second-order double cascade

Two Stages Single Decay




```
directive sample 0.03
directive plot !a; !b; !c

new a@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; ?a; Amp_hi(a,b)

run 1000 of (Amp_lo(a,b) | Amp_lo(b,c))

let A() = !a; A()
run 100 of A()
```

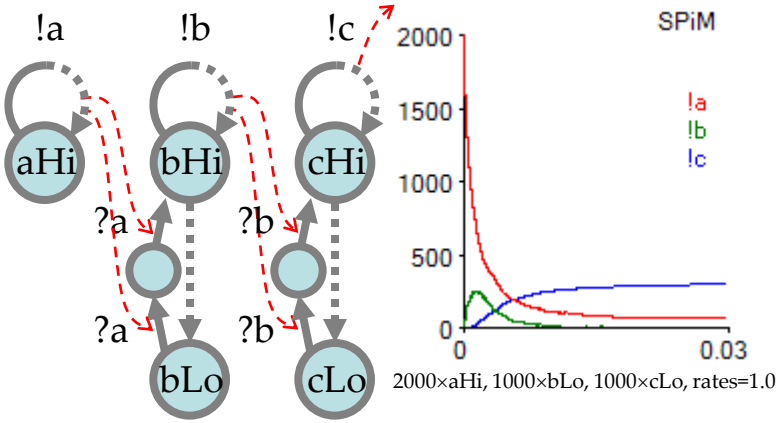
Two Stages Double Decay




```
directive sample 0.03
directive plot !a; !b; !c

new a@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_mi(a,b)
and Amp_mi(a:chan, b:chan) =
  do ?a; Amp_hi(a,b) or delay@1.0; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; Amp_mi(a,b)

run 1000 of (Amp_lo(a,b) | Amp_lo(b,c))

let A() = !a; A()
run 100 of A()
```

```
directive sample 0.03
directive plot !a; !b; !c

new a@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_mi(a,b)
and Amp_mi(a:chan, b:chan) =
  do ?a; Amp_hi(a,b) or delay@1.0; Amp_lo(a,b)
and Amp_lo(a:chan, b:chan) =
  ?a; Amp_mi(a,b)

run 1000 of (Amp_lo(a,b) | Amp_lo(b,c))

let A() = !a; A()
run 100 of A()
```

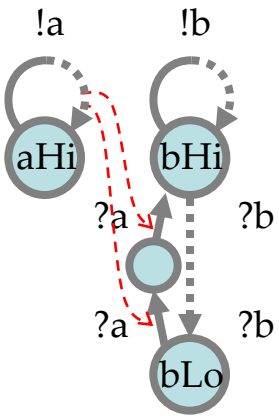
Fig 27. Zero Order Double Cascade

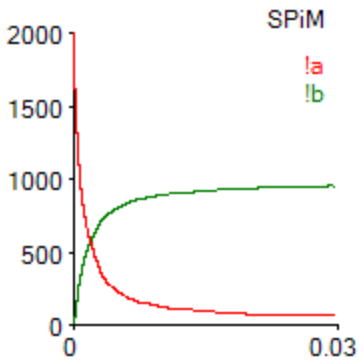



```
directive sample 0.03
directive plot !a; !b; !c

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

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

run 1000 of (Amp_lo(a,b) | Amp_lo(b,c))

let A() = !a; delay@1.0; A()
run 2000 of A()
```

Fig. 27. Zero-order double cascade

One Stage of Zero-order Double Cascade





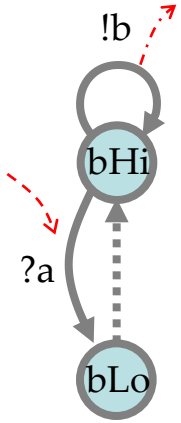

```
directive sample 0.03
directive plot !a; !b

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

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

run 1000 of Amp_lo(a,b)

let A() = !a; delay@1.0; A()
run 2000 of A()
```

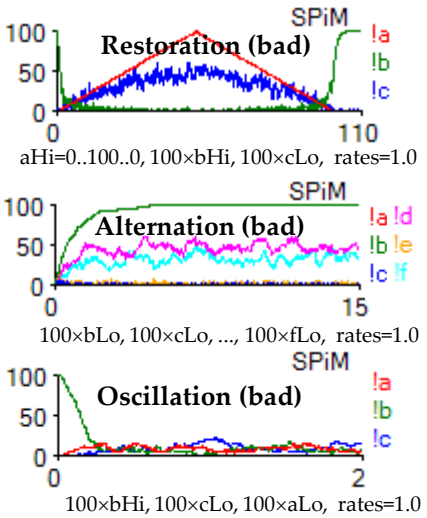
Section 3.8

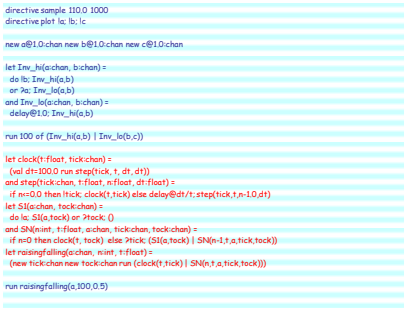
Fig 28. Simple Inverter

$aHi + bHi \rightarrow^{\gamma} aHi + bLo$

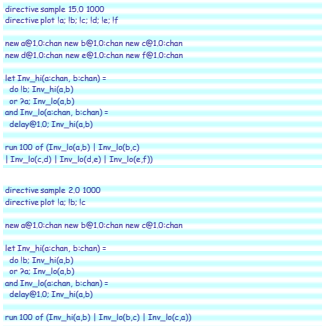
$bLo \rightarrow bHi$







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

new a@1.0:chan new b@1.0:chan new c@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) | Inv_lo(b,c))

let clock(t:float, tick:chan) =
  (val dt=100.0 run step(tick, t, dt, dt))
and step(tick:chan, t:float, n:float, dt:float) =
  if n<=0.0 then !tick; clock(t,tick) else delay@dt/t; step(tick,t,n-1.0,dt)
let SI(a:chan, tock:chan) =
  do !a; SI(a,tock) or ?tock; ()
and SN(n:int, t:float, a:chan, tick:chan, tock:chan) =
  if n=0 then clock(t, tock) else ?tick; (SI(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)
```



```
directive sample 15.0 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

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_lo(a,b) | Inv_lo(b,c)
| Inv_lo(c,d) | Inv_lo(d,e) | Inv_lo(e,f))

directive sample 2.0 1000
directive plot !a; !b; !c

new a@1.0:chan new b@1.0:chan new c@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) | Inv_lo(b,c) | Inv_lo(c,a))
```

Fig. 28. Simple inverter

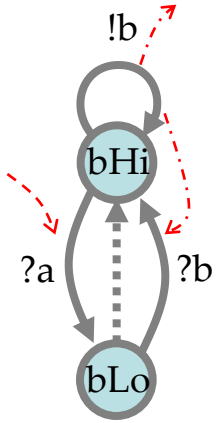
$[bHi]^* = -\gamma[aHi][bHi] + [bLo]$

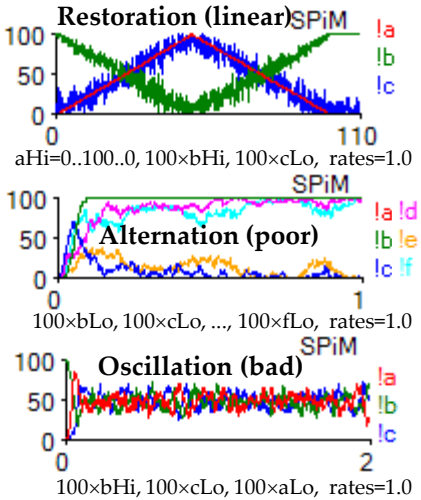
assume $[bHi]^* = 0$
$[bLo] = \gamma[aHi][bHi]$

$[bHi] = Max - [bLo] = Max - \gamma[aHi][bHi]$
$[bHi](1+\gamma[aHi]) = Max$
$[bHi] = Max/(1+\gamma[aHi])$

Fig 29. Feedback Inverter

$aHi + bHi \xrightarrow{\gamma} aHi + bLo$
$bLo + bHi \xrightarrow{\gamma} bHi + bHi$
$bLo \rightarrow bHi$





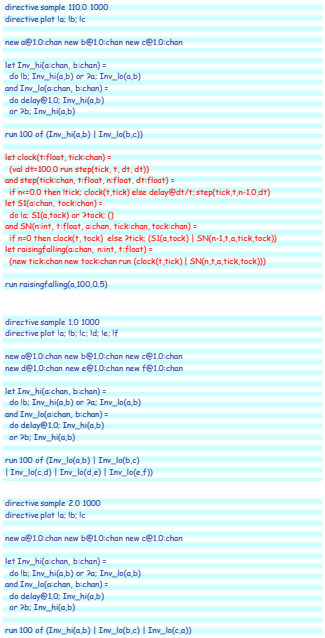


Fig. 29. Feedback inverter

$[bHi]^* = -\gamma[aHi][bHi] + \gamma[bLo][bHi] + [bLo]$

assume $[bHi]^* = 0$
$\gamma[aHi][bHi] = \gamma[bLo][bHi] + [bLo]$
$\gamma[aHi][bHi] = \gamma[bLo]([bHi] + 1/\gamma)$
$[bLo] = [aHi][bHi]/([bHi] + 1/\gamma) \sim [aHi]$

$[bHi] = Max - [bLo] \sim Max - [aHi]$

Fig 30. Double-Height Inverter

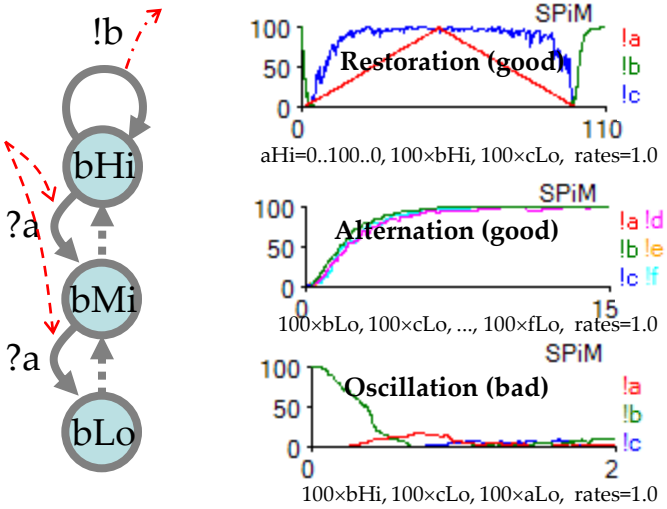


Fig. 30. Double-height inverter

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

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

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

let clock(t:float, tick:chan) =
  (val dt=100.0 run step(tick, t, dt, dt))
and step(tick:chan, t:float, n:float, dt:float) =
  if n=0.0 then !tick; clock(t,tick) else delay@dt/t; step(tick,t,n-1.0,dt)
let S1(a:chan, tock:chan) =
  do !a; S1(a,tock) or ?tock; ()
and 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)
```

```
directive sample 15.0 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

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

run 100 of (Inv2_lo(a,b) | Inv2_lo(b,c)
| Inv2_lo(c,d) | Inv2_lo(d,e) | Inv2_lo(e,f))
```

```
directive sample 2.0 1000
directive plot !a; !b; !c

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

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

Fig 31. Double Height Feedback Inverter

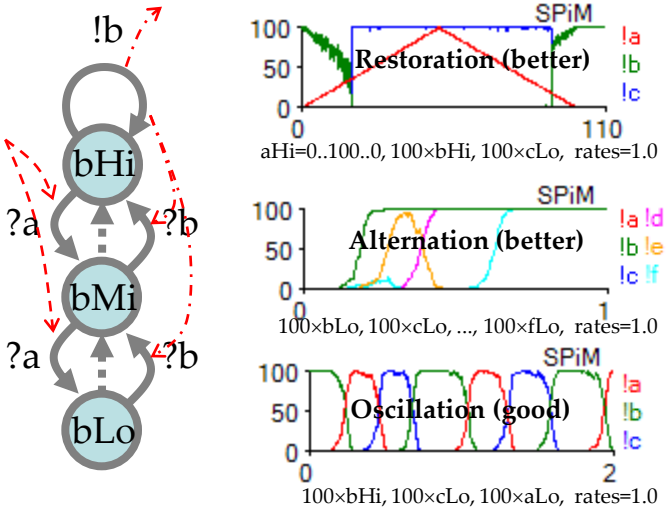


Fig. 31. Double-height feedback

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

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

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

let clock(t:float, tick:chan) =
  (val dt=100.0 run step(tick, t, dt, dt))
and step(tick:chan, t:float, n:float, dt:float) =
  if n<=0.0 then !tick; clock(t,tick) else delay@dt/t; step(tick,t,n-1.0,dt)
let SI(a:chan, tock:chan) =
  do !a; SI(a,tock) or ?tock; ()
and SN(nint, t:float, a:chan, tick:chan, tock:chan) =
  if n=0 then clock(t, tock) else ?tick; (SI(a,tock) | SN(n-1,t,a,tick,tock))
let raisingfalling(a:chan, nint, t:float) =
  (new tick:chan new tock:chan run (clock(t,tick) | SN(n,t,a,tick,tock)))

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

```
directive sample 1.0 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

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

run 100 of (Inv2_lo(a,b) | Inv2_lo(b,c)
| Inv2_lo(c,d) | Inv2_lo(d,e) | Inv2_lo(e,f))
```

```
directive sample 2.0 1000
directive plot !a; !b; !c

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

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

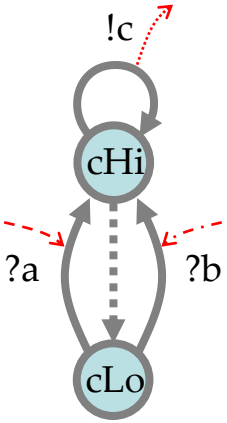
with parameter scaling:

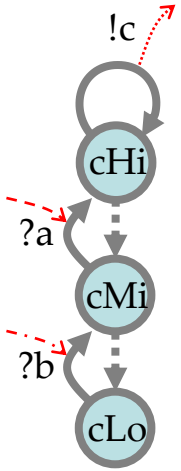
```
directive sample 2.0 1000
directive plot !a; !b; !c
val scale = 10.0

new a@1.0/scale:chan new b@1.0/scale:chan new c@1.0/scale: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 delay@1.0; Inv2_hi(a,b)
  or ?a; Inv2_lo(a,b)
  or ?b; Inv2_hi(a,b)
and Inv2_lo(a:chan, b:chan) =
  do delay@1.0; Inv2_mi(a,b)
  or ?b; Inv2_mi(a,b)

let many(n:float) =
(if n<=2.0 then () else (many(n-1.0) |
  (Inv2_hi(a,b) | Inv2_lo(b,c) | Inv2_lo(c,a))
))

run many(100.0*scale)
```

Section 3.9

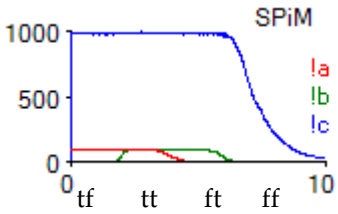
Fig 32. Or/And

$aHi+cLo \rightarrow^{\gamma} aHi+cHi$
$bHi+cLo \rightarrow^{\gamma} bHi+cHi$
$cHi \rightarrow cLo$



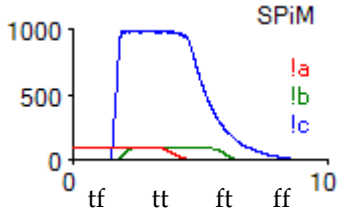


c = a Or b



1000×cLo, rates=1.0

c = a And b

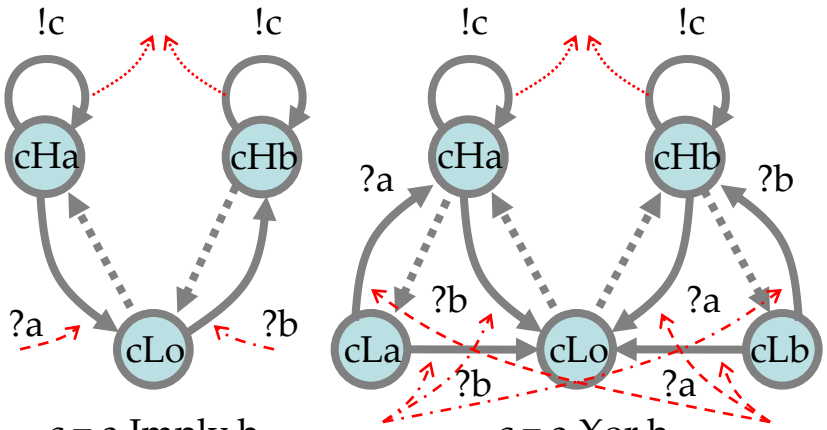


1000×cLo, rates=1.0

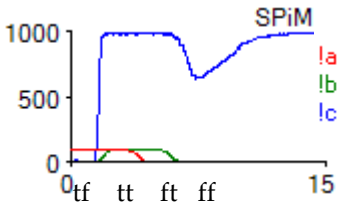
Fig. 32. Or and And

Fig 33. Imply / Xor

$aHi+cHa \rightarrow^{\gamma} aHi+cLo$
$bHi+cLo \rightarrow^{\gamma} bHi+cHb$
$cHb \rightarrow cLo$
$cLo \rightarrow cHa$

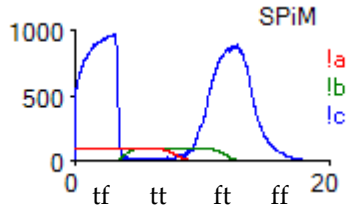


c = a Imply b



1000×cLo, rates=1.0

c = a Xor b



500×cLa, 500×cLb, rates=1.0

Fig. 33. Imply and Xor

Section 3.10

Fig 34. Memory Elements

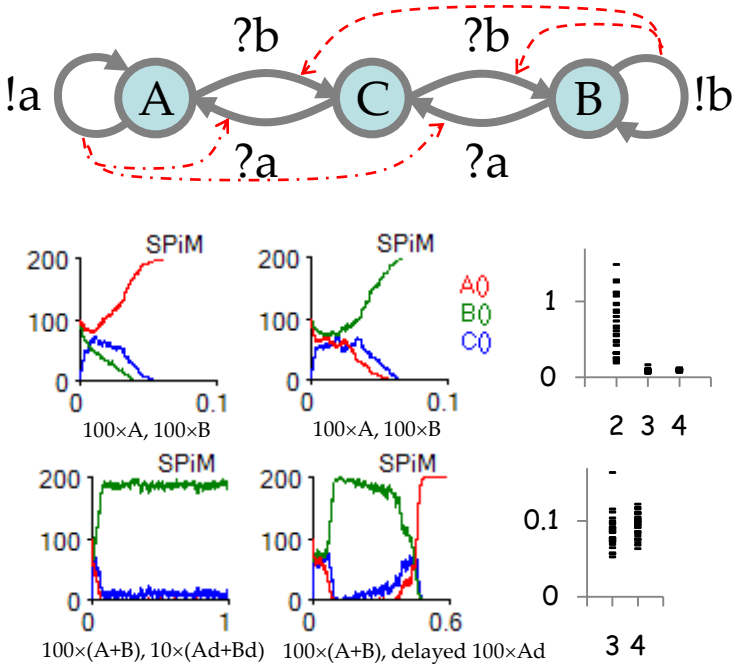
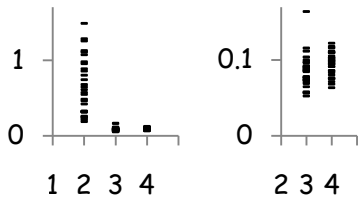
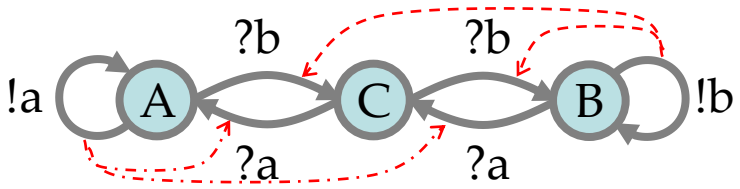


Fig. 34. Memory elements

Memory Elements

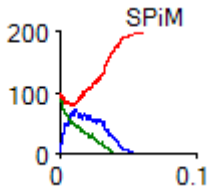


Fast Switching

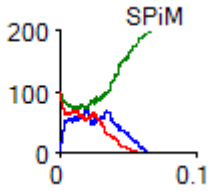


Bistability

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

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

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

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



100×A, 100×B

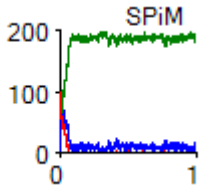


100×A, 100×B

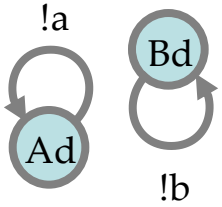
A()
B()
C()

Noise Resistance

```
directive sample 1.0
directive plot A(); B(); C()

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

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

let Ad() = !a; Ad()
and Bd() = !b; Bd()

run 100 of (A() | B())
run 10 of (Ad() | Bd())
```



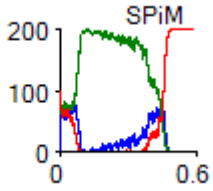
100×(A+B), 10×(Ad+Bd)



Resettability

After the bistable has settle on 100 green, a signal is injected that forces it to red. The strength of the injected signal is 100, the same strength as the bistable signal.

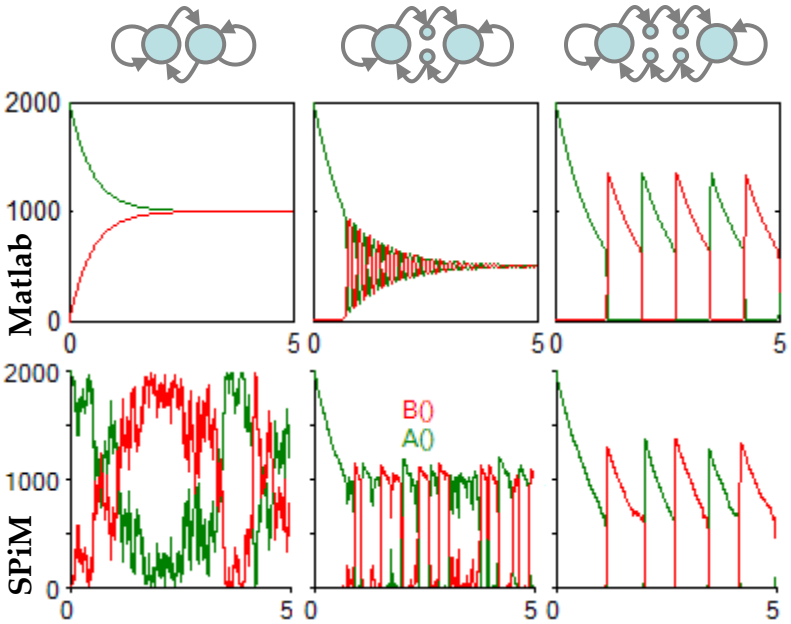
```
directive sample 0.6
directive plot A(); B(); C()

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

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

let Ad() = !a; Ad()

run 100 of (A() | B())
run 100 of delay@10.0; delay@10.0;
delay@10.0; delay@10.0;
delay@10.0; Ad()
```



100×(A+B), delayed 100×Ad

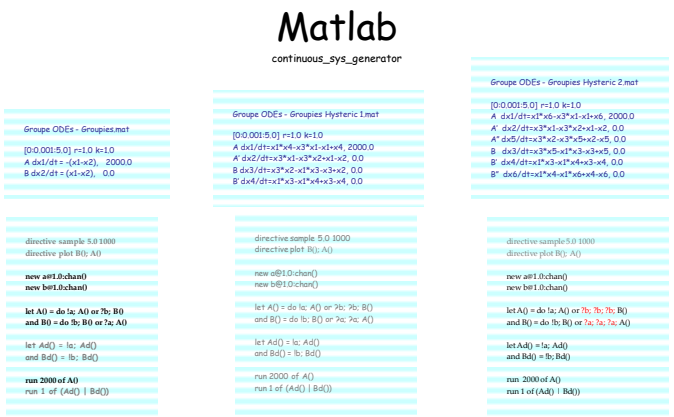
Section 3.11

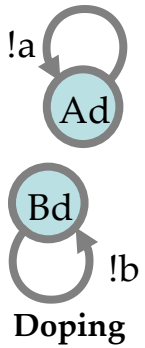
Fig 35. Continuous vs. Discrete Groupies



2000×A , 0×B , 1×A_d , 1×B_d , r = 1.0

Fig. 35. Discrete vs. continuous modeling





Section 4

Fig 36. Polyautomata Reactions

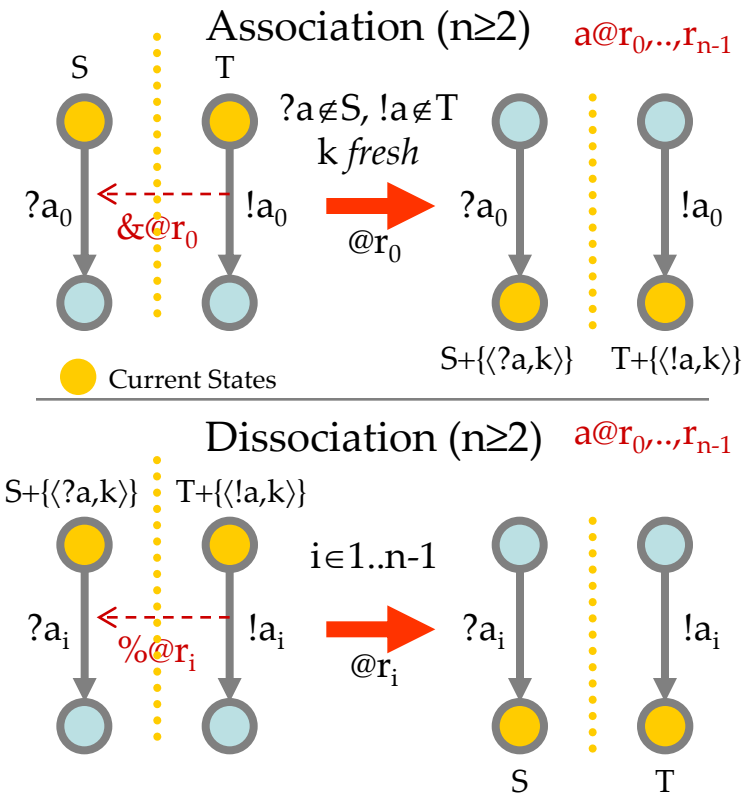
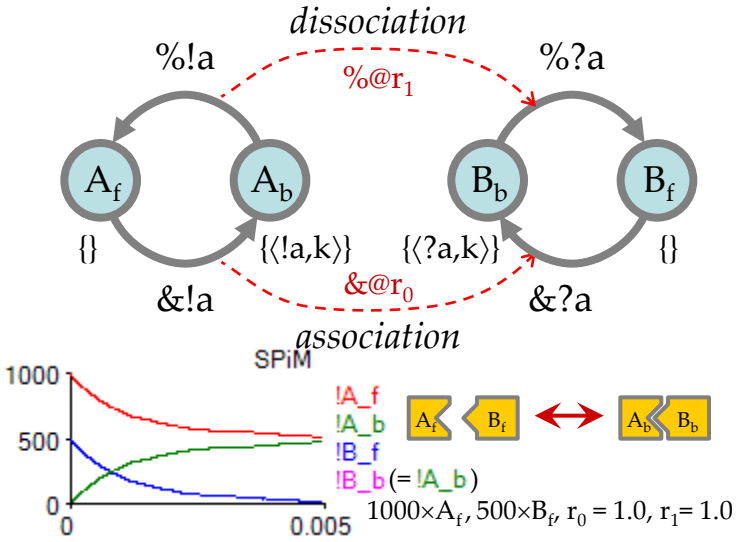


Fig. 36. Polyautomata reactions

Fig 37. Complexation/decomplexation




```
directive sample 0.005
directive plot !A_f; !A_b; !B_f; !B_b
new A_f:chan new A_b:chan new B_f:chan new B_b:chan

val mu = 1.0    val lam = 1.0
new a@mu:chan(chan)

let Af() = (new k@lam:chan run do !a(k); Ab(k) or !A_f)
and Ab(k:chan) = do !k; Af() or !A_b

let Bf() = do ?a(k); Bb(k) or !B_f
and Bb(k:chan) = do ?k; Bf() or !B_b

run (1000 of Af() | 500 of Bf())
```

Fig. 37. Complexation/decomplexation

Fig 38. Enzymatic reactions

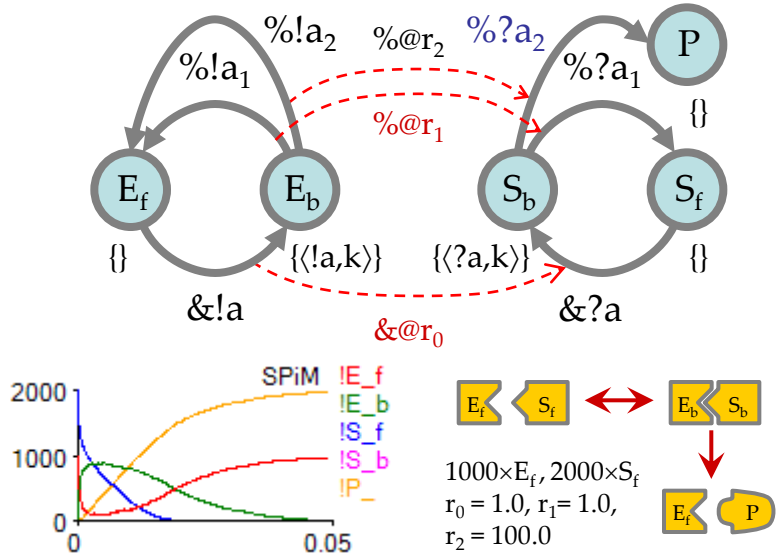


Fig. 38. Enzymatic reactions

```
directive sample 0.05 1000
directive plot !E_f; !E_b; !S_f; !S_b; !P_
new E_f:chan new E_b:chan new S_f:chan new S_b:chan
new P_:chan

val r0 = 1.0    val r1 = 1.0    val r2 = 100.0
new a@r0:chan(chan,chan)

let P() = !P_

let Ef() =
  (new k1@r1:chan new k2@r2:chan
   run do !a(k1,k2); Eb(k1,k2) or !E_f)
and Eb(k1:chan,k2:chan) =
  do !k1; Ef() or !k2; Ef() or !E_b

let Sf() = do ?a(k1,k2); Sb(k1,k2) or !S_f
and Sb(k1:chan,k2:chan) =
  do ?k1; Sf() or ?k2; P() or !S_b

run (1000 of Ef() | 2000 of Sf())
```

Fig 39. Homodimerization

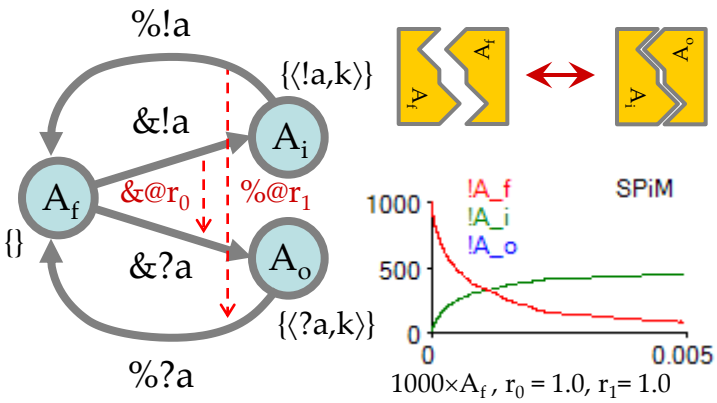


Fig. 39. Homodimerization

```
directive sample 0.005 10000
directive plot !A_f; !A_i; !A_o
new A_f:chan new A_i:chan new A_o:chan

new a@1.0:chan(chan)

let Af() =
  (new k@1.0:chan
   run do ?a(k); Ai(k) or !a(k); Ao(k) or !A_f)

and Ai(k:chan) = do ?k; Af() or !A_i

and Ao(k:chan) = do !k; Af() or !A_o

run 1000 of Af()
```

Fig 40. Bidirectional polymerization

```
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())
```

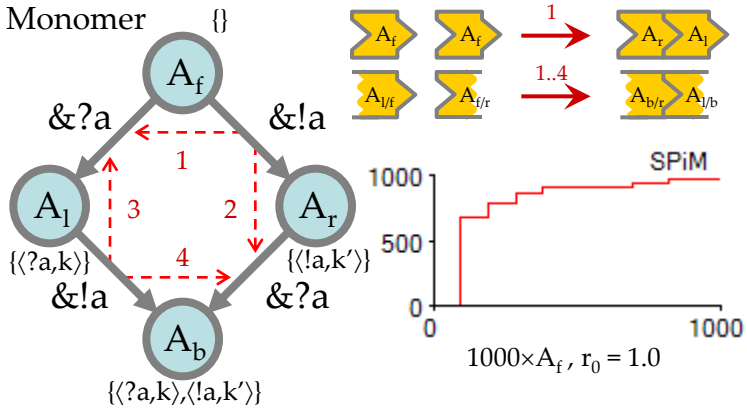
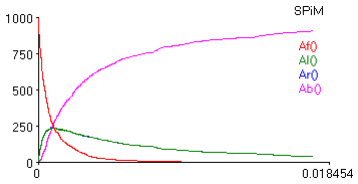


Fig. 40. Bidirectional polymerization




```
directive sample 1000.0
directive plot ?count
(* directive plot Af(); Al(); Ar(); Ab() *)

type Link = chan(chan)
type Barb = chan

val lam = 1000.0 (* set high for better counting; it is not used
for dissociation *)
val mu = 1.0
new c@mu:chan(Link)
new enter@lam:chan(Barb)
new count@lam:Barb

let Af() =
  (new rht@lam:Link run
   do !c(rht); Ar(rht)
   or ?c(lft); Al(lft))

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

and Ar(rht:Link) =
  ?c(lft); Ab(lft,rht)

and Ab(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) =
  (val dt=100.0 run step(tick, t, dt, dt))
and step(tick:chan, t:float, n:float, dt:float) =
  if n<=0.0 then !tick; clock(t,tick) else delay@dt/t; step(tick,t,n-1.0,dt)

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

run 1000 of Af()
run (clock(100.0, tick) | Scan())
```

Fig 41. Automata Polymers

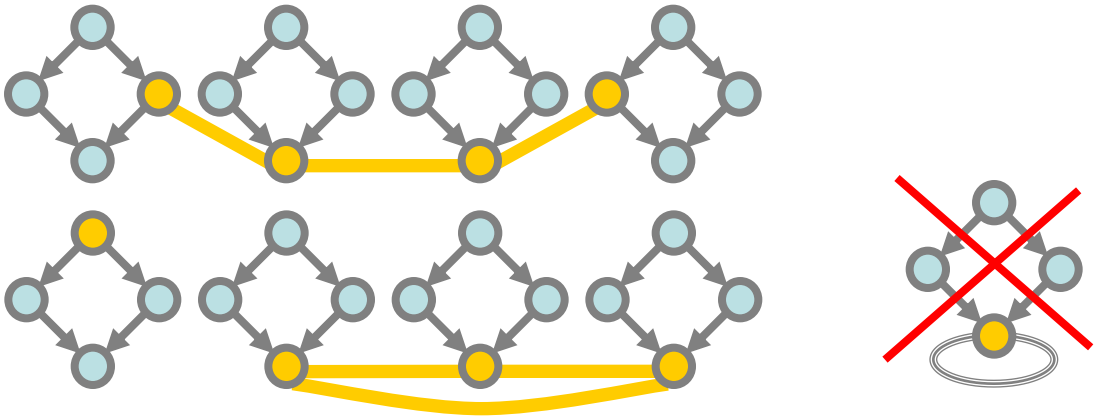


Fig. 41. Automata polymers

Fig 42. Actin-like polymerization

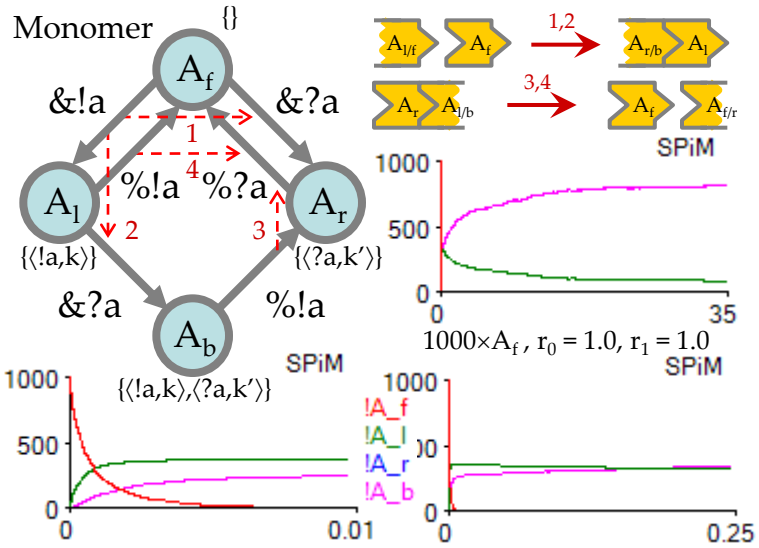


Fig. 42. Actin-like polymerization

```
directive sample 0.01 (* 0.25, 35.0 *) 1000
directive plot !A_f; !A_l; !A_r; !A_b
new  A_f:chan new A_l:chan new A_r:chan new A_b:chan

val lam = 1.0 (* dissoc *)
val mu = 1.0 (* assoc *)
new c@mu:chan(chan)

let Af() =
  (new lft@lam:chan run
   do !c(lft); Al(lft)
   or ?c(rht); Ar(rht) or !A_f)

and Al(lft:chan) =
   do !lft; Af()
   or ?c(rht); Ab(lft,rht) or !A_l

and Ar(rht:chan) =
   do ?rht; Af() or !A_r

and Ab(lft:chan, rht:chan) =
   do !lft; Ar(rht) or !A_b

run 1000 of Af()
```

Fig 43. Typical Monomer Interactions

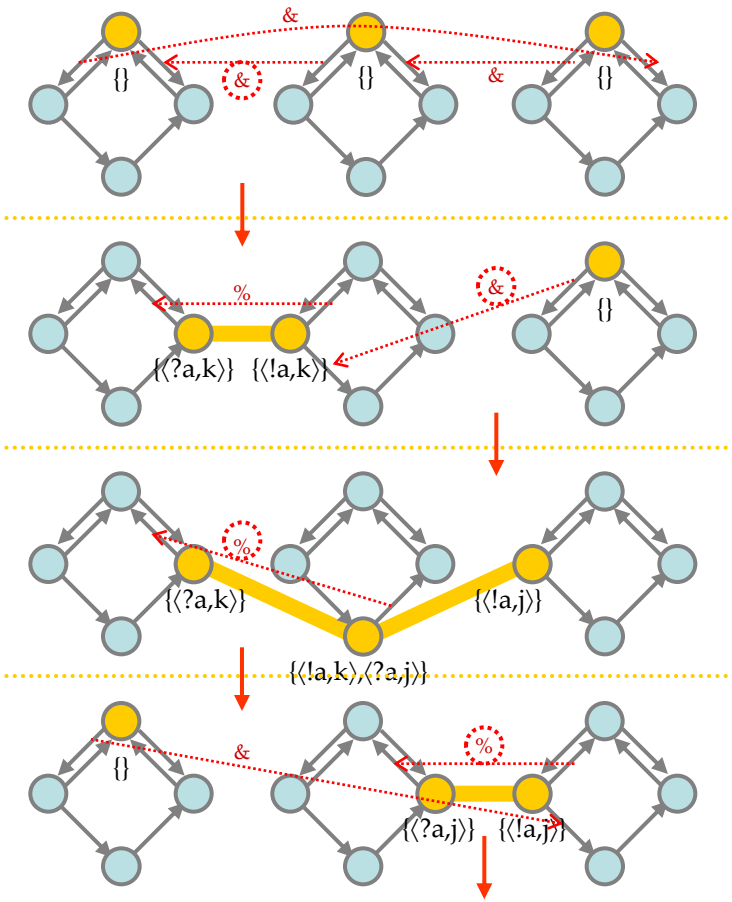


Fig. 43. Typical monomer interactions