Exercises

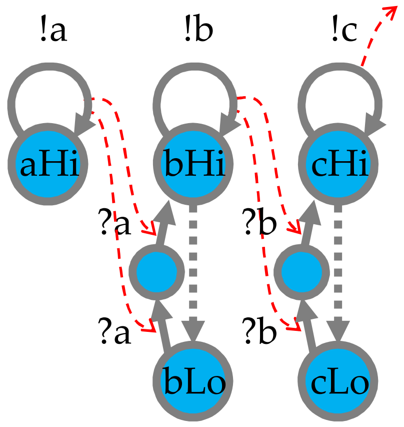
Luca Cardelli

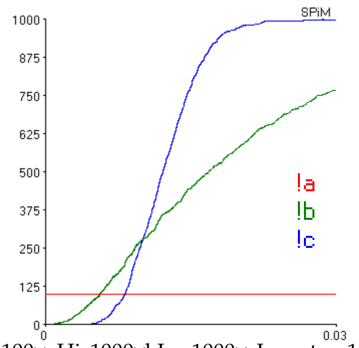
Microsoft Research

Open Lectures for PhD Students in Computer Science
Warsaw 2009-03..05

http://lucacardelli.name

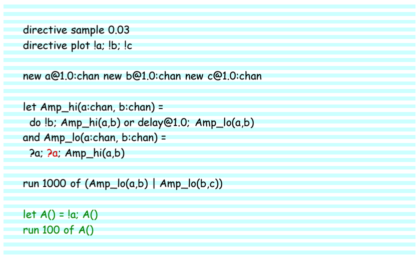
Exercise 1





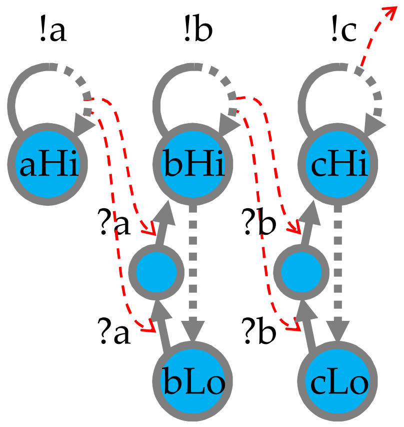
100×aHi, 1000×bLo, 1000×cLo, rates=1.0

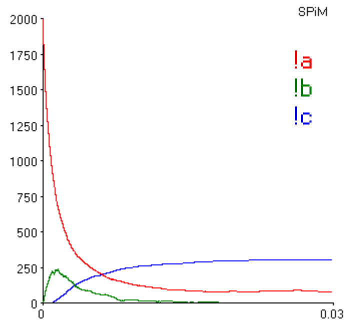
Second-Oder Regime cascade:
a signal amplifier (MAPK)
aHi > 0 ⇒ cHi = max



```
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()
```

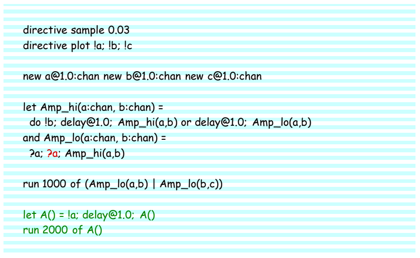
Write these automata in CGF and translate them to chemical reactions.





2000×aHi, 1000×bLo, 1000×cLo, rates=1.0

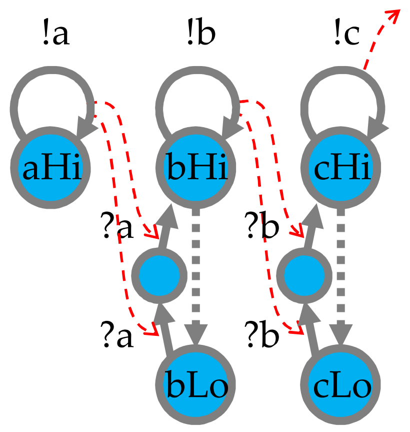
Zero-Oder Regime cascade:
a signal *divider!*
aHi = max ⇒ cHi = 1/3 max



```
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()
```

Exercise 1a Solution

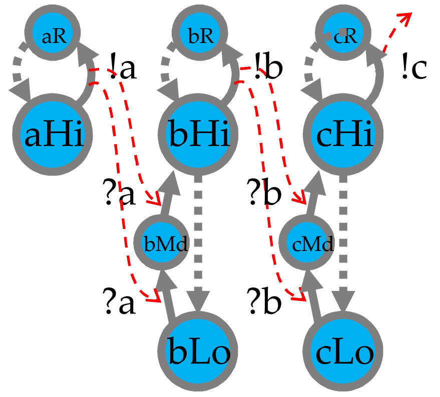
$aHi = !a_{(1.0)};aHi$
$bLo = ?a_{(1.0)};\ bMd$
$bMd = ?a_{(1.0)};\ bHi$
$bHi = !b_{(1.0)};\ bHi \oplus \tau_{(1.0)};\ bLo$
$cLo = ?b_{(1.0)};\ cMd$
$cMd = ?b_{(1.0)};\ cHi$
$cHi = !c_{(1.0)};\ cHi \oplus \tau_{(1.0)};\ cLo$



$aHi + bLo \rightarrow^{1.0} aHi + bMd$
$aHi + bMd \rightarrow^{1.0} aHi + bHi$
$bHi \rightarrow^{1.0} bLo$
$bHi + cLo \rightarrow^{1.0} bHi + cMd$
$bHi + cMd \rightarrow^{1.0} bHi + cHi$
$cHi \rightarrow^{1.0} cLo$

Exercise 1b Solution

$aHi = !a_{(1.0)};aR$
$aR = \tau_{(1.0)};\ aHi$
$bLo = ?a_{(1.0)};\ bMd$
$bMd = ?a_{(1.0)};\ bHi$
$bHi = !b_{(1.0)};\ bR \oplus \tau_{(1.0)};\ bLo$
$bR = \tau_{(1.0)};\ bHi$
$cLo = ?b_{(1.0)};\ cMd$
$cMd = ?b_{(1.0)};\ cHi$
$cHi = !c_{(1.0)};\ cR \oplus \tau_{(1.0)};\ cLo$
$cR = \tau_{(1.0)};\ cHi$



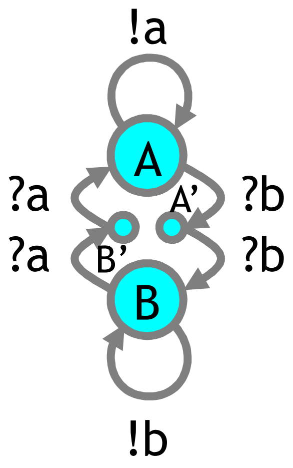
$aHi + bLo \rightarrow^{1.0} aR + bMd$
$aHi + bMd \rightarrow^{1.0} aR + bHi$
$aR \rightarrow^{1.0} aHi$
$bHi \rightarrow^{1.0} bLo$
$bHi + cLo \rightarrow^{1.0} bR + cMd$
$bHi + cMd \rightarrow^{1.0} bR + cHi$
$bR \rightarrow^{1.0} bHi$
$cHi \rightarrow^{1.0} cLo$
$cR \rightarrow^{1.0} cHi$

Note: no reaction from cHi to cR etc. because there is nothing (here) to interact with c.

The chemical system is incomplete (it does not say how cHi would behave in a bigger system), while the automata already specify what would happen (if we remove the red bit in cHi above we obtain the same chemical reactions).

Exercise 2

Q: What does this do?



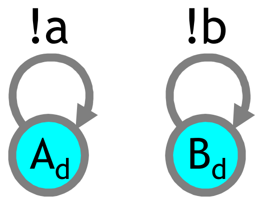
```
directive sample 10.0 1000
directive plot Ga(); Gb()

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())
```

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

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

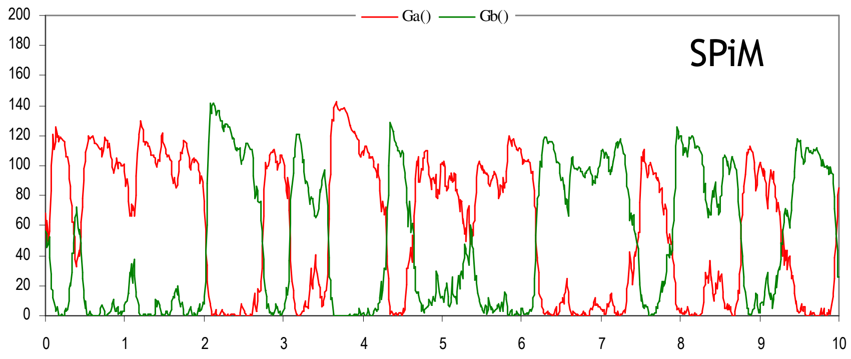
Derive the ODEs from these "Hysteric Groupies" automata. Either by going through the chemical reactions and the Law of Mass Action (easier), or directly from the Process Rate Equation.

Doping

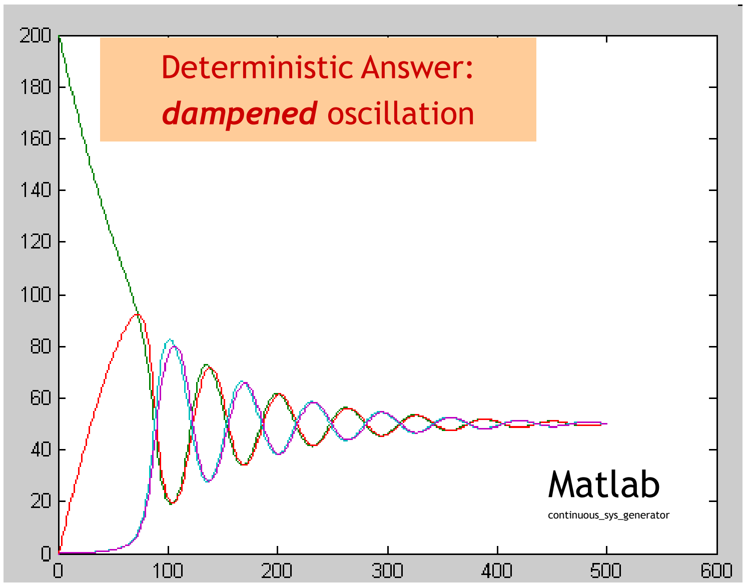


ODE predicts dampened oscillation, while the stochasic system keeps oscillating at max level.

Stochastic Answer:
robust quasi-oscillation

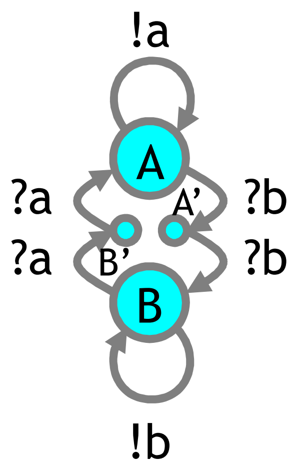


Deterministic Answer:
dampened oscillation



Exercise 2 Solution

Q: What does this do?



```
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())
```

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

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

$$A+B \rightarrow^r A+B' \quad A+B' \rightarrow^r A+A \qquad A+B_d \rightarrow^r A'+B_d \quad A'+B_d \rightarrow^r B+B_d$$

$$B+A \rightarrow^r B+A' \quad B+A' \rightarrow^r B+B \qquad B+A_d \rightarrow^r B'+A_d \quad B'+A_d \rightarrow^r A+A_d$$

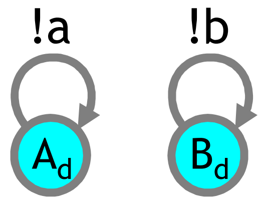
$$\begin{aligned}
d[A]/dt &= r[A][B']-r[B][A]-r[A][B_d]+r[B'][A_d]\\
d[A']/dt &= r[B][A]-r[B][A']+r[A][B_d]-r[A'][B_d]\\
d[B]/dt &= r[B][A']-r[A][B]-r[B][A_d]+r[A'][B_d]\\
d[B']/dt &= r[A][B]-r[A][B']+r[B][A_d]-r[B'][A_d]
\end{aligned}$$

$$d[A_d]/dt = 0 \qquad d[B_d]/dt = 0$$

$$\begin{aligned}
d[A]/dt &= r[A][B']-r[B][A]-rk[A]+rk[B']\\
d[A']/dt &= r[B][A]-r[B][A']+rk[A]-rk[A']\\
d[B]/dt &= r[B][A']-r[A][B]-rk[B]+rk[A']\\
d[B']/dt &= r[A][B]-r[A][B']+rk[B]-rk[B']
\end{aligned}$$

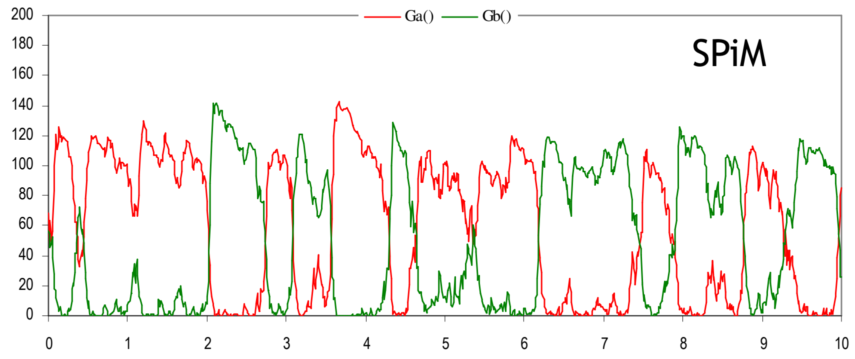
$[A_d]$,$[B_d]$ are constant; assume them both = k

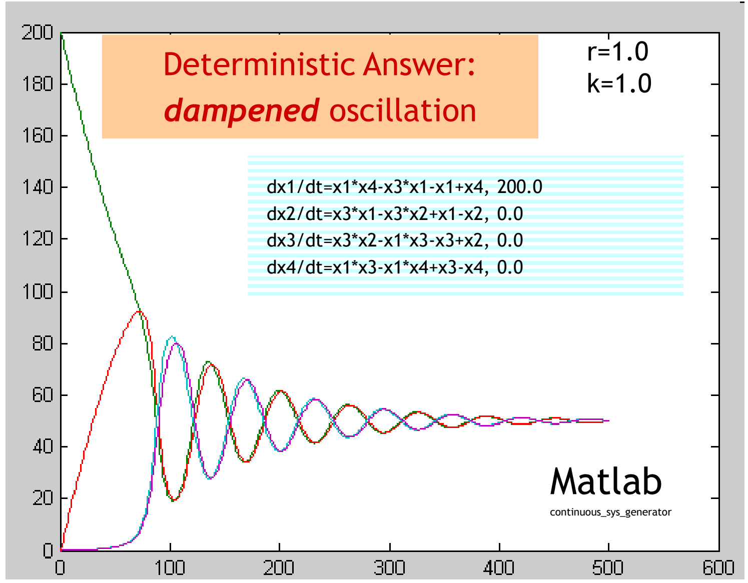
Doping



ODE predicts dampened oscillation, while the stochasic system keeps oscillating at max level.

Stochastic Answer:
robust quasi-oscillation

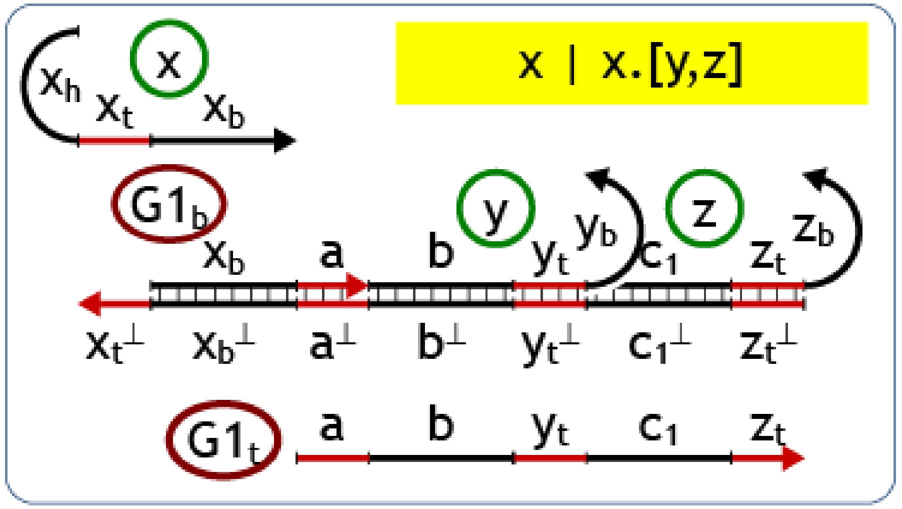




Deterministic Answer:
dampened oscillation

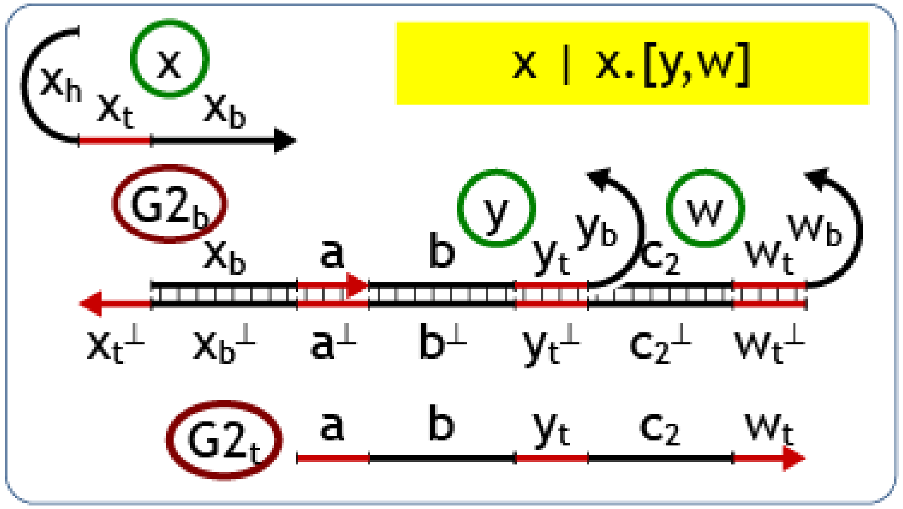
```
dx1/dt=x1*x4-x3*x1-x1+x4, 200.0
dx2/dt=x3*x1-x3*x2+x1-x2, 0.0
dx3/dt=x3*x2-x1*x3-x3+x2, 0.0
dx4/dt=x1*x3-x1*x4+x3-x4, 0.0
```

Exercise 3: x.[y,z] | x.[y,w] Interference



x | x.[y,z]
x
x_h x_t x_b
$G1_b$
x_b a b y y_t y_b z c_1 z_t z_b
$x_t^\perp$ $x_b^\perp$ $a^\perp$ $b^\perp$ $y_t^\perp$ $c_1^\perp$ $z_t^\perp$
$G1_t$
a b y_t c_1 z_t

~~a,b~~,c_1 fresh; x_h generic



x | x.[y,w]
x
x_h x_t x_b
$G2_b$
x_b a b y y_t y_b w c_2 w_t w_b
$x_t^\perp$ $x_b^\perp$ $a^\perp$ $b^\perp$ $y_t^\perp$ $c_2^\perp$ $w_t^\perp$
$G2_t$
a b y_t c_2 w_t

~~a,b~~,c_2 fresh; x_h generic

- Suppose we 'forgot' to take a,b fresh, so they are shared by the two gates. Something goes horribly wrong from these initial conditions:

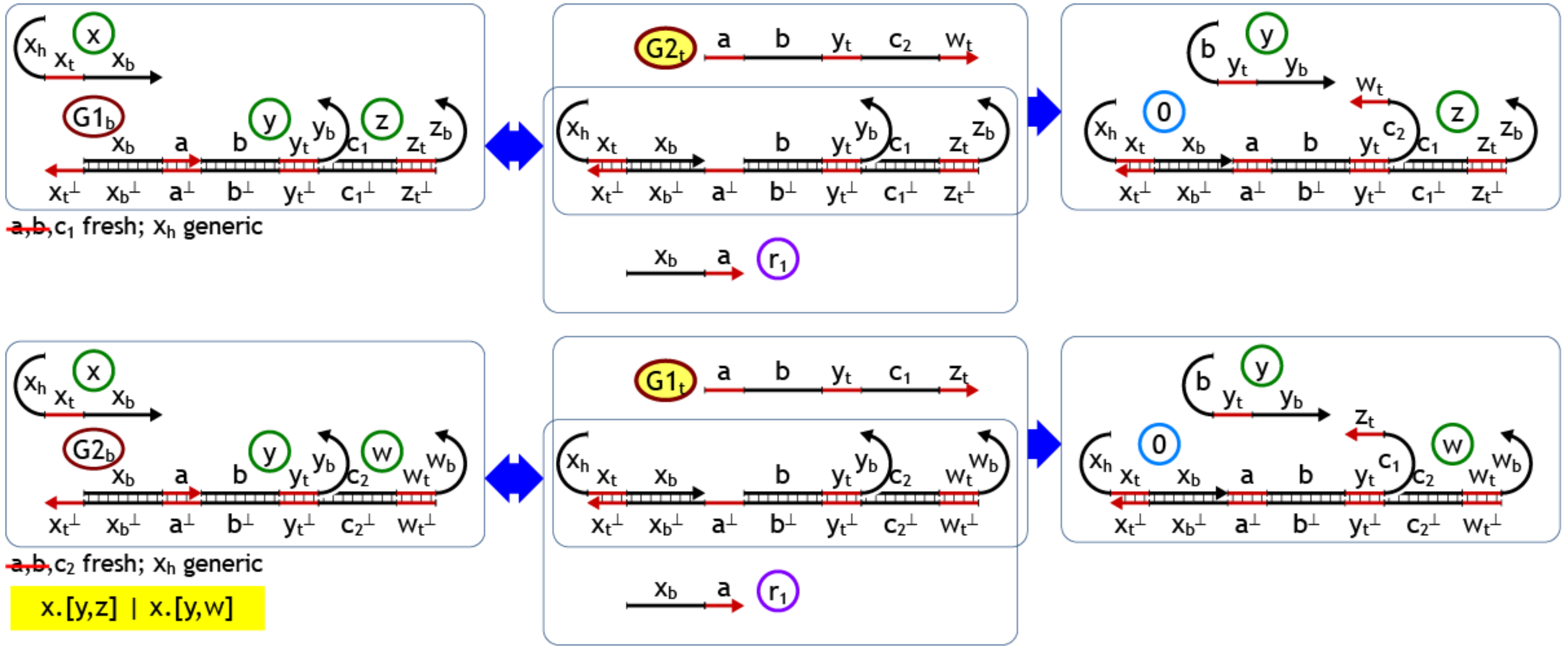
  x | x.[y,z] | x | x.[y,w]

  where x.[y,z] = $G1_b$,$G1_t$ and x.[y,w] = $G2_b$,$G2_t$

- What goes wrong?

Exercise 3 Solution

Deadlocks! Consider x | x | x.[y,z] | x.[y,w], and suppose we had taken c fresh (hence different c_1,c_2), but did *not* used gate-unique segments for a,b:



x_h x_t x x_b
$G1_b$ x_b a b y y_t y_b c_1 z z_t z_b
$x_t^\perp$ $x_b^\perp$ $a^\perp$ $b^\perp$ $y_t^\perp$ $c_1^\perp$ $z_t^\perp$
~~a,b,~~c_1 fresh; x_h generic
$G2_t$ a b y_t c_2 w_t
x_b a r_1
0
x_h x_t x x_b
$G2_b$ x_b a b y y_t y_b c_2 w w_t w_b
$x_t^\perp$ $x_b^\perp$ $a^\perp$ $b^\perp$ $y_t^\perp$ $c_2^\perp$ $w_t^\perp$
~~a,b,~~c_2 fresh; x_h generic
x.[y,z] | x.[y,w]
$G1_t$ a b y_t c_1 z_t

The $G2_t$ trigger can bind to the wrong $G1_b$ backbone and get stuck there, and vice versa, without ever releasing z or w.

This is just a made-up problem, but one must watch out for all kinds of possible interferences.

Exercise 4: x.y.z | [x,y].w Interference

[David Soloveichik]

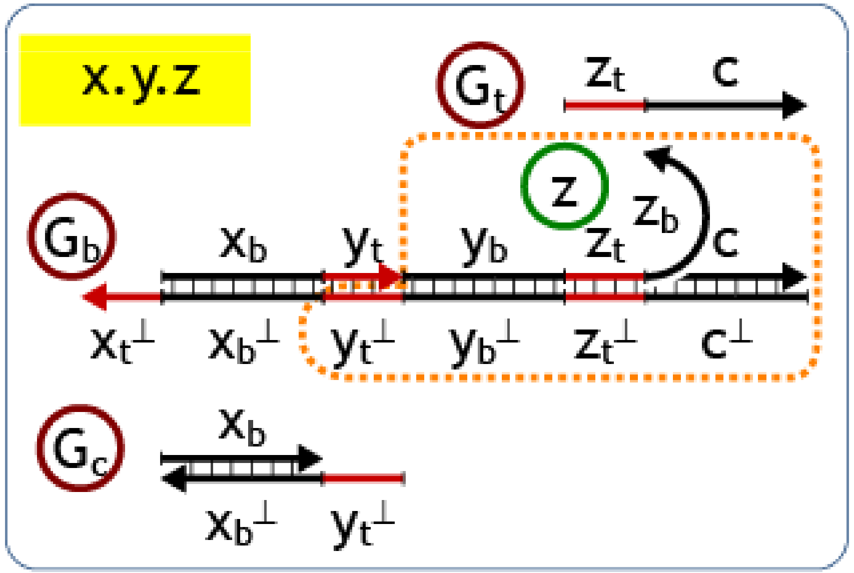
Consider curried gates without the a,b segments (example below): instead of releasing x_b,a and b,y_t segments, they would release x_b,y_t.

But that is exactly the strand r_1 of an [x,y].w gate: the strand that reverts the x input. This definitely causes an interference between x.y.z and [x,y].w.

Find a situation where the presence (x.y.z as below) or absence (x.y.z as in previous slide) of this interference causes different outcomes.

Hint: it changes outcome *probability*.

Note: the a,b segments prevent the interference.



c fresh; x_h,y_h generic
(without the a,b segments)

Exercise 4 Solution

[David Soloveichik]

Consider curried gates without the a,b segment; instead of releasing x_b,a and b,y_t segments, they would release x_b,y_t.

But that is exactly the segment r_1 of the [x,y].z gate; the one that reverts the x input. And x.y.z has a an x_b,y_t collector that will remove r_1, and make x bind to [x,y].z irreversibly! That's not supposed to happen (in absence of y).

This interference would not change the logic of the gates, but it would change their stochastic behavior.

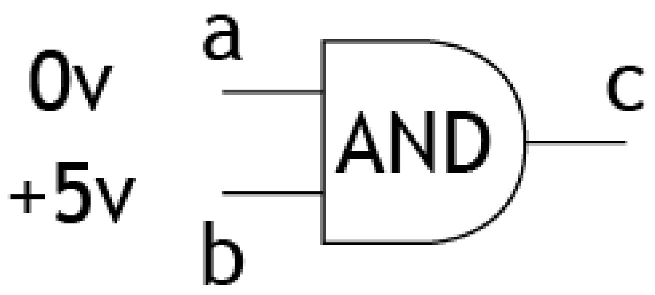
Consider: x | x.y.z | [x,y].w. Without interference, x can bind only reversibly to [x,y].w, and irreversibly to x.y.z. Hence with high probability this would produce y.z | [x,y].w, and if *later* providing y it would produce z with high probability.

However, with the r_1 interference, x would initially bind equally likely to [x,y].w (and x.y.z), and the x_b,y_t could be removed by the x.y.z collector. Hnece x would irreversibly bind equally likely to [x,y].w and x.y.z, and if *later* providing y, we would now get z or w with equal probability.

The extra a,b segments break the contiguity of x_b,y_t, avoiding the interference.

Exercise 5: Boolean Networks

Boolean Networks to Strand Algebra

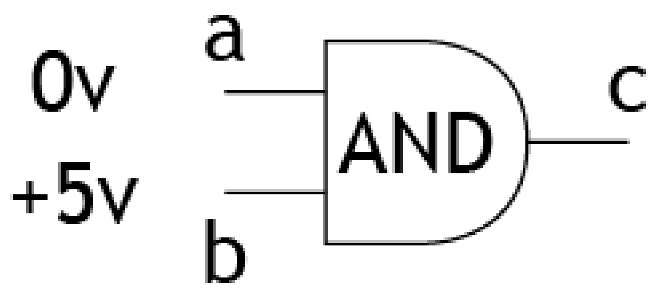


Find an encoding of Boolean networks in Strand Algebra.

It’s enough to show how to encode and AND gate that takes Boolean signals on a,b wires and produces a Boolean signal on the c wire, and a NOT gate. (Their combination, a NAND gate, is a universal gate.)

Exercise 5 Solution

Boolean Networks to Strand Algebra



$([a_F, b_F].c_F)^*$ |
$([a_F, b_T].c_F)^*$ |
$([a_T, b_F].c_F)^*$ |
$([a_T, b_T].c_T)^*$ |
a_F | b_T

Represent each wire in the system (e.g. ‘a’) as a pair of signals (‘a_T’,’a_F’). Represent each Boolean gate by its truth table, with a join for each truth table entry. Map initial voltages to initial signals.

NOT gate is similar: $(a_F.b_T)^*$ | $(a_T.b_F)^*$

Exercise 6: Wet Vending Machine Controller

A coffee vending machine controller, Vend, accepts two coins for coffee; an ok is given after the first coin and then either a second coin (for coffee) or an abort (for refund) is accepted:

```
Vend = ?coin. ![ok,mutex]. (Coffee | Refund)
Coffee = ?[mutex,coin]. !coffee. (Coffee | Vend)
Refund = ?[mutex,abort]. !refund. (Refund | Vend)
```

Exercise: compile that to the Combinatorial Strand Algebra; if you do it by the U(P) algorithm you can then heavily hand-optimize it.

Each Vend iteration spawns two branches, Coffee and Refund, waiting for either coin or abort. The branch not taken in the mutual exclusion is left behind; this could skew the system towards one population of branches. Therefore, when the Coffee branch is chosen and the system is reset to Vend, we also spawn another Coffee branch to dynamically balance the Refund branch that was not chosen; conversely for Refund.

Standard questions can be asked: what happens if somebody inserts three coins very quickly? Or somebody presses refund twice? Etc.

Exercise 6 Solution

Two hand-optimized solutions:

$Vend$ |
$([Vend, coin].[ok, mutex, Coffee, Refund])^*$ |
$([Coffee, mutex, coin].[coffee, Vend, Coffe])^*$ |
$([Refund, mutex, abort].[refund, Vend, Refund])^*$

This second solution however removes the ‘branch balancing’ effect:

$Vend$ |
$([Vend, coin].[ok, mutex])^*$ |
$([mutex, coin].[coffee, Vend])^*$ |
$([mutex, abort].[refund, Vend])^*$

The algorithmic solution, with some optimization, should be [Marek Cygan]:

```
Y | Z | T |
(Y.[Y,Y2] | [Y2, Vend].Y3 | [Y3,coin].[ok,mutex,Coffee,Refund])*
(Z.[Z,Z2] | [Z2, Coffee].Z3 | [Z3,mutex,coin].[coffee,Coffee,Vend])*
(T.[T,T2] | [T2, Refund].T3 | [T3,mutex,abort].[refund,Refund,Vend])*
```