Genetic Networks in Stochastic π-Calculus

Luca Cardelli

Microsoft Research
Cambridge UK

2004-08-26

www.luca.demon.co.uk

Genetic Networks

Strange facts about genetic networks:

The output of each gate is fixed and pre-determined; it is never a function of the input! (Except maybe the quantity of output is a function of the quantity of input.) This is not a functional or operator calculus.

There are inputs and anti-inputs (inhibition). An anti-input is not the same as absence of an input: it will block the gate in a specific way. Inhibition is widespread. This is not term-rewriting, nor Petri nets.

Feedback is widespread: e.g. each gate can send input to itself. This requires an asynchronous communication model to avoid immediate self-deadlocks. In particular, even the simplest gates cannot be modeled as a single synchronous process. This is not Communicating Sequential Processes.

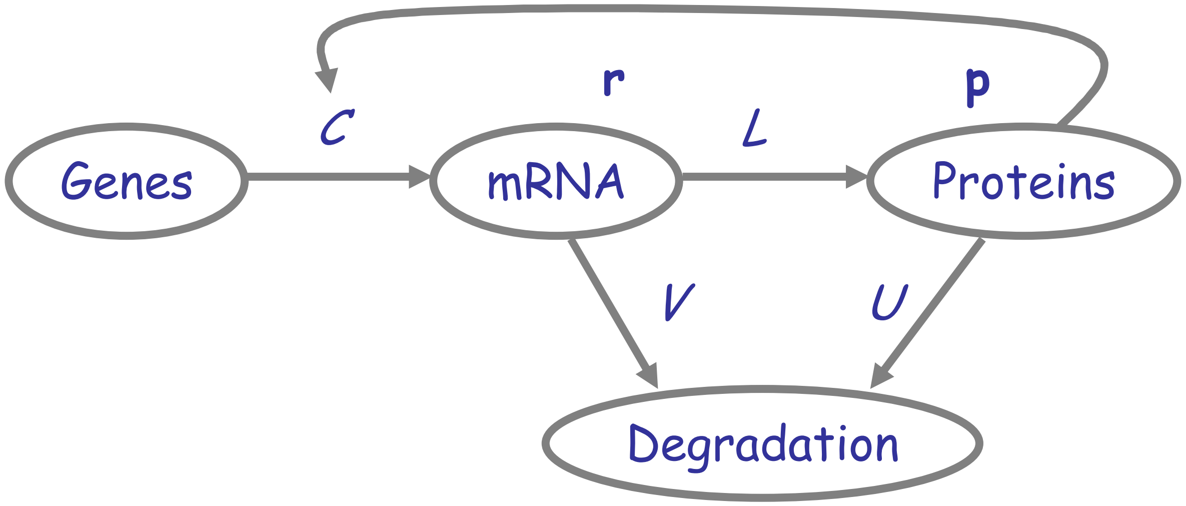
Messages themselves have behavior (e.g., they will stochastically decay, and this is a fundamental property), hence messages should really be modeled as processes as well. This is not message-passing.

The apparently crude idea of broadcasting a whole bunch of asynchronous decaying messages to activate a future gate is subtly clever: it means there are never any "pipeline full" deadlocks propagating backwards, even in presence of abundant feedback loops. Any attempt to use data-flow-style modeling of these circuits seems doomed because of loops. This is not data-flow.

The amount of output can be a function of the amount of input, but it is not clear how important this really is in many cases. Gene circuits are robust w.r.t. radical changes in concentrations (e.g. during cell growth), and in many cases seem to switch digitally (with steep sigmoids). The combination of degradation and concentration-based interaction rates can produce reasonably stable, normalized, signals in a noisy environment.

(The Classical Approach)

[Chen, He, Church]



$$\frac{d\mathbf{r}}{dt} = f(\mathbf{p}) - V\mathbf{r}$$

$$\frac{d\mathbf{p}}{dt} = L\mathbf{r} - U\mathbf{r}$$

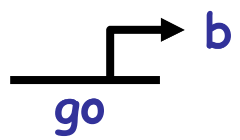
n: number of genes
r mRNA concentrations (n-dim vector)
p protein concentrations (n-dim vector)

$f(\mathbf{p})$ transcription functions:
(n-dim vector polynomials on **p**)

Nullary Gate

Let's begin by modeling transcription factors as simple messages.

Nullary Gate



$go[b] \triangleq \tau_{\varepsilon}.\ (b_r\langle\rangle \mid go[b])$

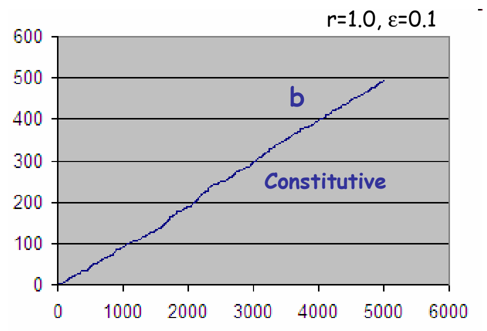
(recursive, parametric) process definition

stochastic delay (τ) of constitutive transcription

stochastic rates

gene product, spawn out

and repeat

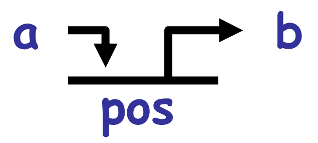


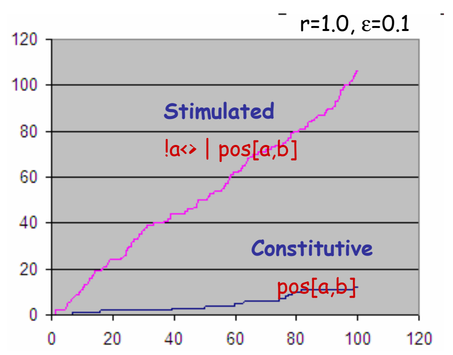
cf.: $go[b] \triangleq b_{\varepsilon}\langle\rangle.\ go[b]$

Not good for two reasons:
- we want to use r as the rate for the binding of b, not for the production of b.
- does not extend to unary and binary gates, where we want a τ in front.

Unary Gates

Designed to handle self-loops!



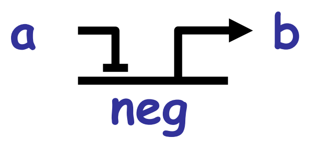


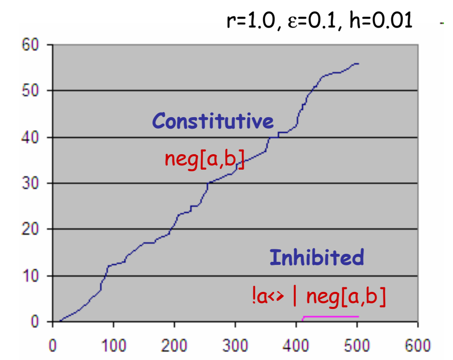
stimulated transcription

or constitutive transcription to always get things started

pars, not dots, to handle self-loops without deadlock

$\mathrm{pos}[a,b] \triangleq$
$a_r().\ (b_s\langle\rangle \mid \mathrm{pos}[a,b])\ +$
$\tau_\varepsilon.\ (b_s\langle\rangle \mid \mathrm{pos}[a,b])$





inhibition delay

$\mathrm{neg}[a,b] \triangleq$
$a_r().\ \tau_h.\ \mathrm{neg}[a,b]\ +$
$\tau_\varepsilon.\ (b_s\langle\rangle \mid \mathrm{neg}[a,b])$

Not my first attempt!

N.B. inputs and outputs are one-to-one. We could put in an amplification factor, but a similar effect can usually be obtained by adjusting the production rates or the persistence rates: see later.

Degradation

Product degradation is extremely important in general; it changes unbounded growth into (roughly) stable signals.

protein producer

$$ptn[p] \triangleq p_r<>.ptn[p] + dk_\delta<>$$

protein product
(can add transcription delay here)

protein degradation site

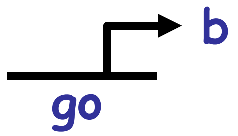
protein interaction site

degradation infrastructure

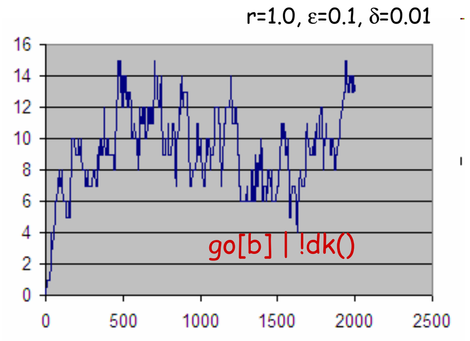
$$!\ dk_\delta()$$

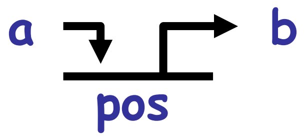
degradation rate

call to producer

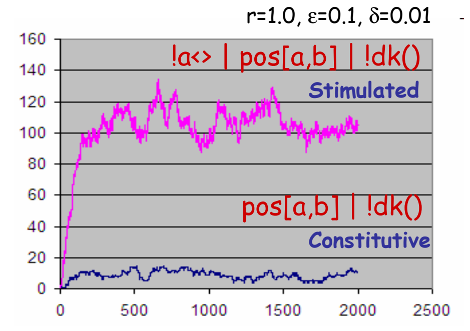


$$go[b] \triangleq \tau_\varepsilon.\ (ptn[b]\ |\ go[b])$$



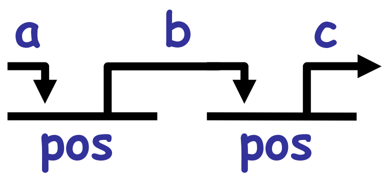


$$pos[a,b] \triangleq$$
$$a_r().\ (ptn[b]\ |\ pos[a,b])\ +$$
$$\tau_\varepsilon.\ (ptn[b]\ |\ pos[a,b])$$



Non-Linear Response

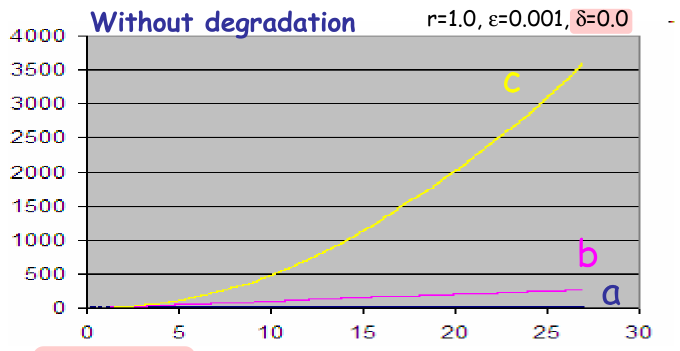
pos[a,b] | pos[b,c]



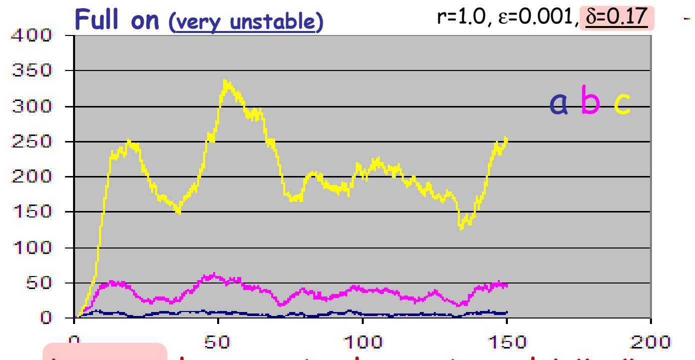
pos[a,b] $\triangleq$
$a_r()$. (ptn[b] | pos[a,b]) +
τ_ε. (ptn[b] | pos[a,b])

ptn[p] $\triangleq$ p_r<>.ptn[p] + dk_δ<>

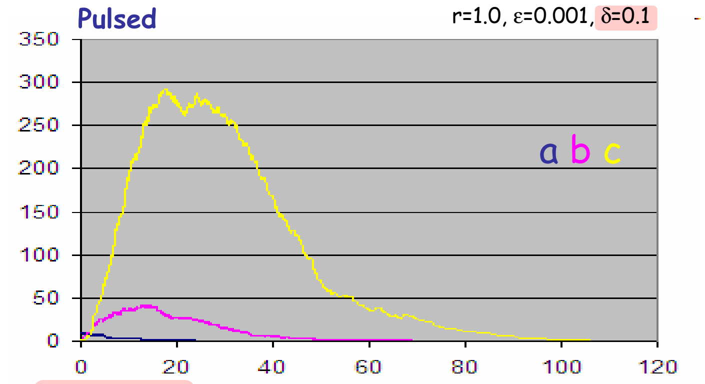
E.g. 1 a that interacts twice before decay can produce 2 b that each interact twice before decay, which produce 4 c…



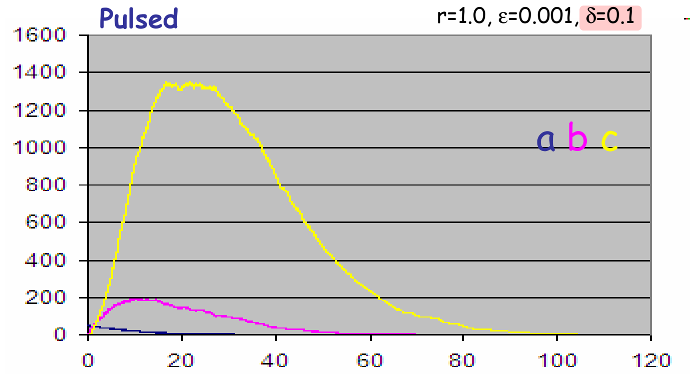
|^{10}ptn[a] | pos[a,b] | pos[b,c]



!ptn[a] | pos[a,b] | pos[b,c] | !dk_δ()



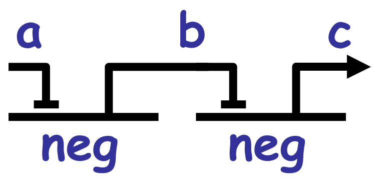
|^{10}ptn[a] | pos[a,b] | pos[b,c] | !dk_δ()



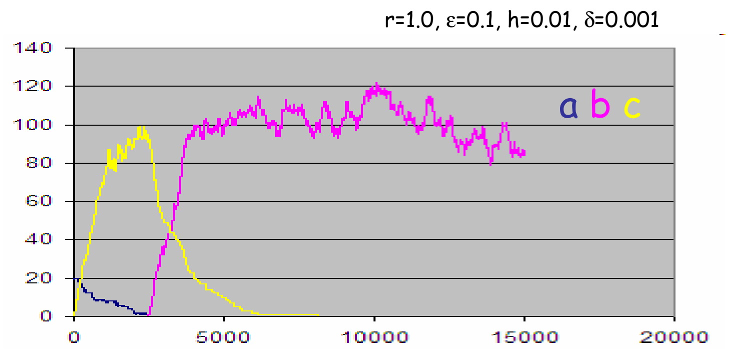
|^{50}ptn[a] | pos[a,b] | pos[b,c] | !dk_δ()

Signal Normalization

neg[a,b] |
neg[b,c]



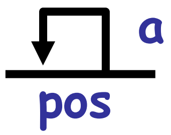
The input level (a), whether weak or strong, is renormalized to a standard level (c).



$|^{20}\text{ptn}[a]$ | neg[a,b] | neg[b,c] | $!\text{dk}_{\delta}()$

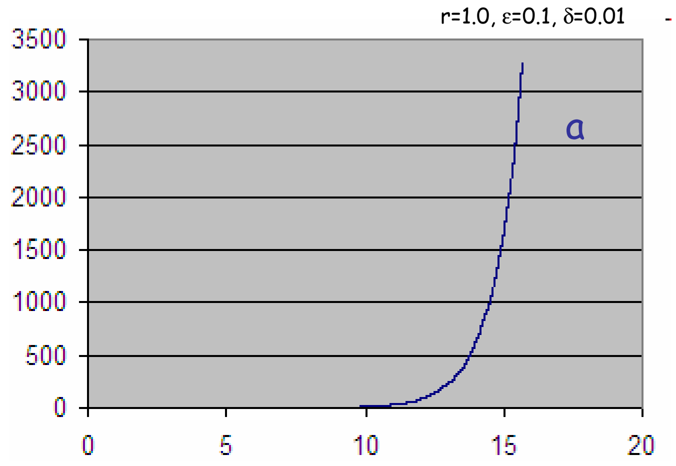
Self Feedback Circuits

pos[a,a]



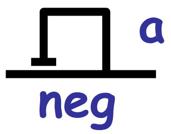
$\text{pos}[a,b] \triangleq$
$\quad a_r().\ (\text{ptn}[b] \mid \text{pos}[a,b]) +$
$\quad \tau_\varepsilon.\ (\text{ptn}[b] \mid \text{pos}[a,b])$

$\text{ptn}[p] \triangleq p_r\langle\rangle.\text{ptn}[p] + \text{dk}_\delta\langle\rangle$



$\text{pos}[a,a] \mid !\text{dk}_\delta()$

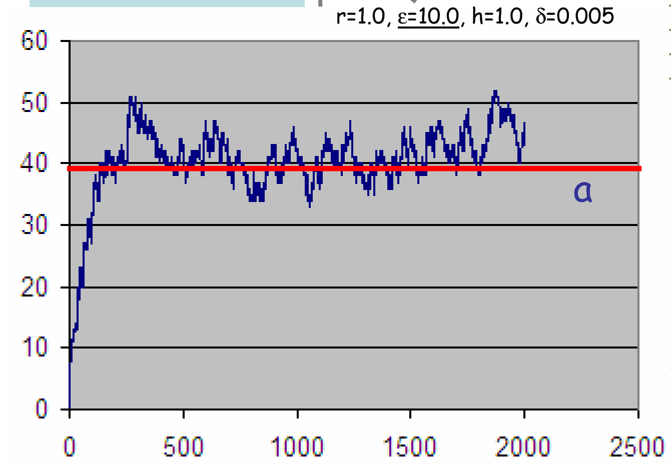
neg[a,a]

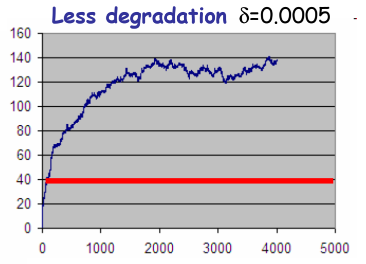


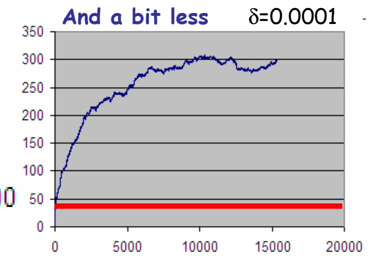
$\text{neg}[a,b] \triangleq$
$\quad a_r().\ \tau_h.\ \text{neg}[a,b] +$
$\quad \tau_\varepsilon.\ (\text{ptn}[b] \mid \text{neg}[a,b])$

$\text{ptn}[p] \triangleq p_r\langle\rangle.\text{ptn}[p] + \text{dk}_\delta\langle\rangle$

high, to raise the signal



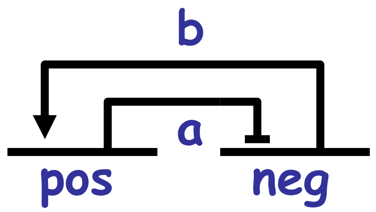




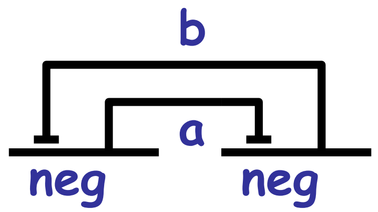
$\text{neg}[a,a] \mid !\text{dk}_\delta()$

Two-gate Feedback Circuits

pos[b,a] | neg[a,b]

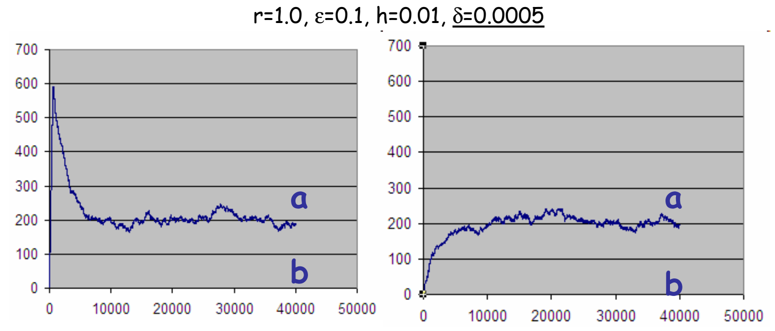


neg[b,a] | neg[a,b]



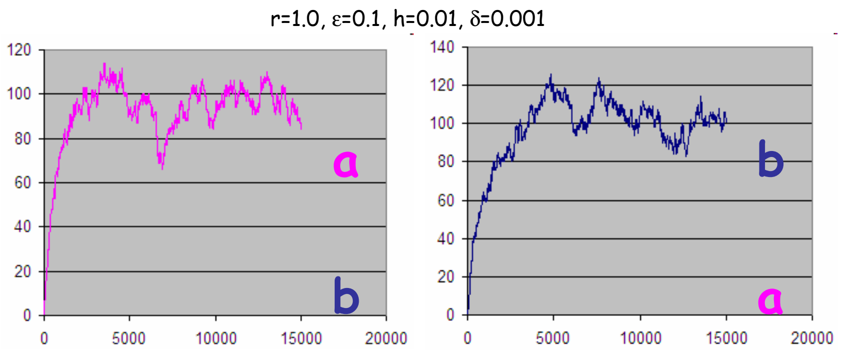
N.B. unlike the neg-self feedback loop, this circuit does not require high-ε gates. It is a model of transcription-translation.

For some degradation rates is quite stable:



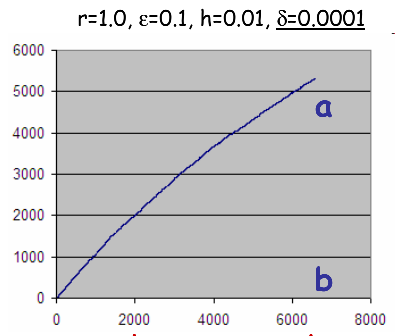
pos[b,a] | neg[a,b] | !dk$_\delta$()

Bistable:



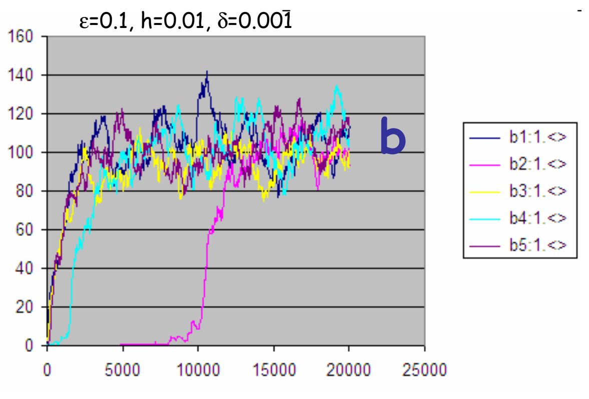
neg[b,a] | neg[a,b] | !dk$_\delta$()

But with a small change in degradation, it goes wild:



A hint for D052?

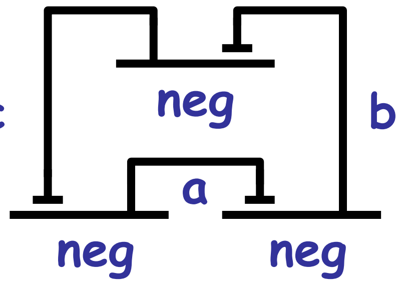
pos[b,a] | neg[a,b] | !dk$_\delta$()



5 runs with r(a)=0.1, r(b)=1.0 shows that circuit is now biased towards expressing b

Repressilator

neg[a,b] |
neg[b,c] |
neg[c,a]



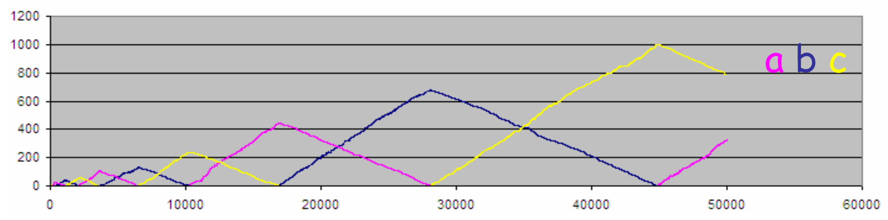
$neg[a,b] \triangleq$
$a_r().\ \tau_h.\ neg[a,b]\ +$
$\tau_\varepsilon.\ (ptn[b]\ |\ neg[a,b])$

$!\ dk_\delta()$

Same circuit, three different degradation models:

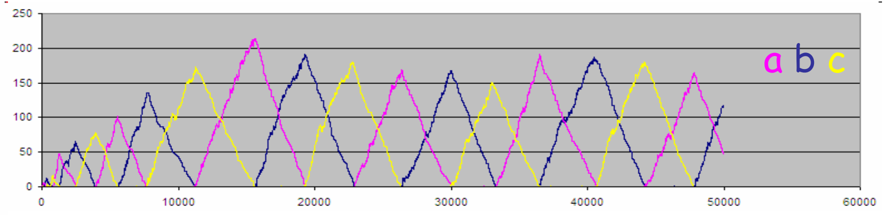
$ptn[p] \triangleq p_r\langle\rangle$

r=1.0, ε=0.1, h=0.04



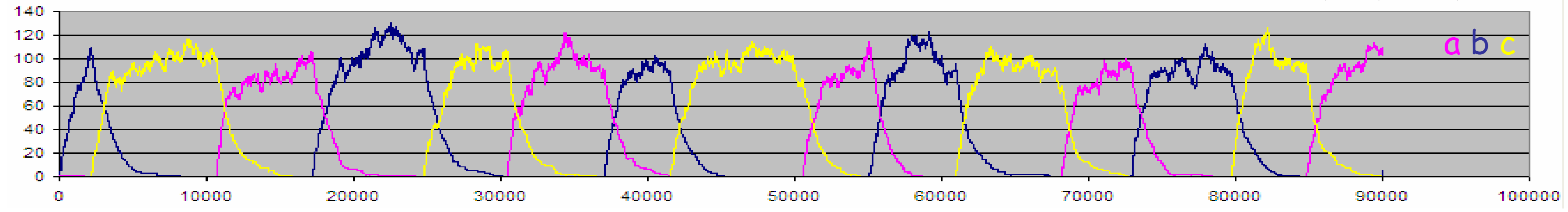
$ptn[p] \triangleq p_r\langle\rangle + dk_\delta\langle\rangle$

r=1.0, ε=0.1, h=0.04, δ=0.0001



$ptn[p] \triangleq p_r\langle\rangle.ptn[p] + dk_\delta\langle\rangle$

r=1.0, ε=0.1, h=0.001, δ=0.001



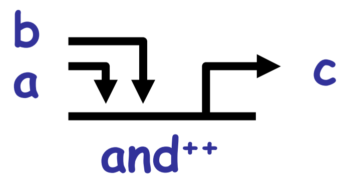
Subtle… at any point one gate is inhibited and the other two can fire constitutively. If one of them fires first, nothing really changes, but if the other one fires first, then the cycle progresses.

Repressilator in SPiM

```
new ptn:<<>>                  (* Protein *)
new dk:0.001:<>               (* Decay rate *)

new neg:<<>,<>>               (* Neg Gate *)
new tInh:0.001:<>             (* Inhibition rate *)
new tCst:0.1:<>               (* Constitutive rate *)

(* Protein-Gene interactions *)
new a:1.0:<> new b:1.0:<> new c:1.0:<>

( !ptn(p); (p<>;ptn<p>+dk<>;())
| !dk()

| !neg(a,b);
    (a(); (tInh(); neg<a,b>) +
     tCst(); (ptn<b> | neg<a,b>))
| !tCst<> | !tInh<>

(* The circuit *)
| neg<a,b> | neg<b,c> | neg<c,a>
)
```

Binary Gates

And-gate with two positive inputs



unbinding delay of first input

$\text{and}^{++}[a,b,c] \triangleq$
$\quad a_r().\ (\tau_u.\text{and}^{++}[a,b,c] + b_s().(c_t\langle\rangle \mid \text{and}^{++}[a,b,c])) +$
$\quad b_s().\ (\tau_u.\text{and}^{++}[a,b,c] + a_r().(c_t\langle\rangle \mid \text{and}^{++}[a,b,c])) +$
$\quad \tau_\varepsilon.(c_t\langle\rangle \mid \text{and}^{++}[a,b,c])$

Here we could model collaborative binding by increasing the binding rate of the second input. Moreover, the strength of the second binding could differ depending on which input is bound first..

$\text{and}^{++}[a,b,c] \triangleq$
$\quad a_r().b_r().(c_t\langle\rangle \mid \text{and}^{++}[a,b,c]) +$
$\quad b_r().a_r().(c_t\langle\rangle \mid \text{and}^{++}[a,b,c]) +$
$\quad \tau_\varepsilon.(c_t\langle\rangle \mid \text{and}^{++}[a,b,c])$

<= Wrong: we cannot ignore stochastic unbinding of inputs, otherwise a spurious input would become a persistent state so that a second spurious input at a much later time would trigger the gate. (In the case of unary gates, we could factor unbinding into the binding time.) Hence the solution above . But then notice that inputs can now really get LOST, so we need to generate enough of them to keep things going.

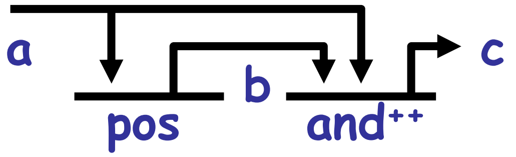
$\text{pos}[a,b] \triangleq$
$\quad a_r().\ (\tau_u.\text{pos}[a,b] + (b_s\langle\rangle \mid \text{pos}[a,b])) +$
$\quad \tau_\varepsilon.\ (b_s\langle\rangle \mid \text{pos}[a,b])$

<= A revised unary gate, for consistency with binary gates?

Feed Forward Loop

$\mathrm{pos}[a,b] \triangleq$
$a_r().(\mathrm{rna}[b] \mid \mathrm{pos}[a,b]) +$
$\tau_\varepsilon.(\mathrm{rna}[b] \mid \mathrm{pos}[a,b])$

$\mathrm{pos}[a,b] \mid$
$\mathrm{and}^{++}[a,b,c]$



$\mathrm{and}^{++}[a,b,c] \triangleq$
$a_r().\ (\tau_u.\mathrm{and}^{++}[a,b,c] + b_r().(\mathrm{rna}[c] \mid \mathrm{and}^{++}[a,b,c])) +$
$b_r().\ (\tau_u.\mathrm{and}^{++}[a,b,c] + a_r().(\mathrm{rna}[c] \mid \mathrm{and}^{++}[a,b,c])) +$
$\tau_\varepsilon.(c_r\langle\rangle \mid \mathrm{and}^{++}[a,b,c])$

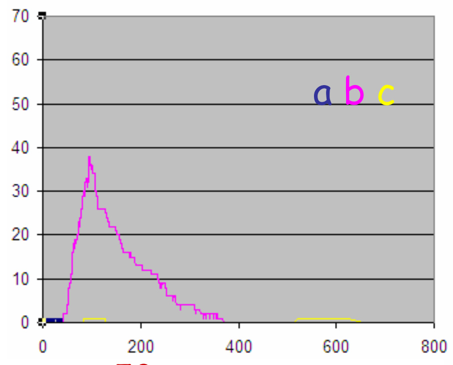
transcription delay (not rate)

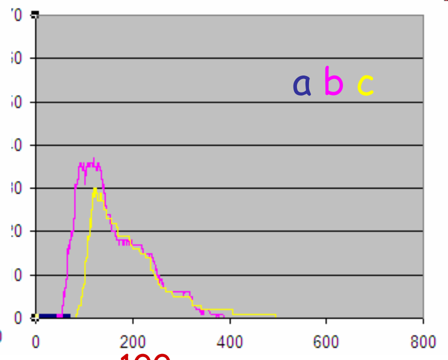
$\mathrm{rna}[p] \triangleq \Delta_{tran}.\mathrm{ptn}[p]$

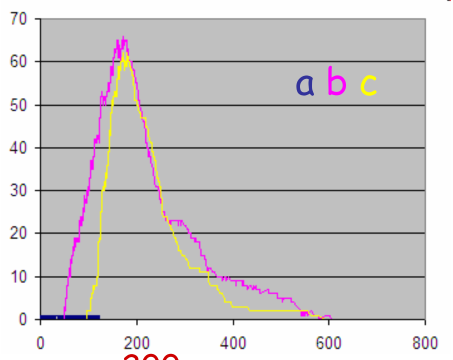
$\mathrm{ptn}[p] \triangleq p_r\langle\rangle.\mathrm{ptn}[p] + dk_\delta\langle\rangle$

$!\ dk_\delta()$

r=1.0, ε=0.0001, δ=0.01, u=0.1, tran=50.0



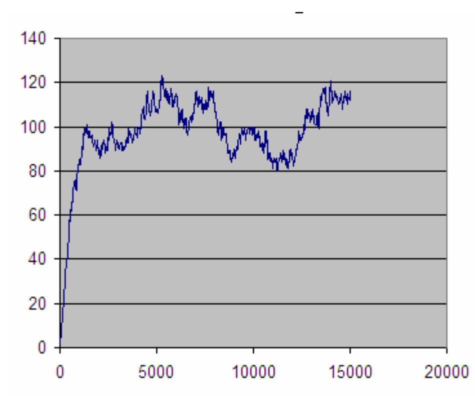




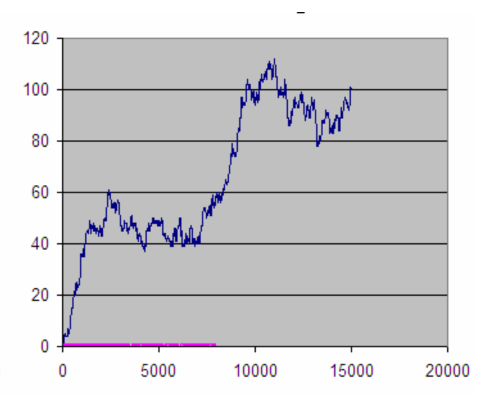
stimuli

$a\langle\rangle;^{50}$ $a\langle\rangle;^{100}$ $a\langle\rangle;^{200}$

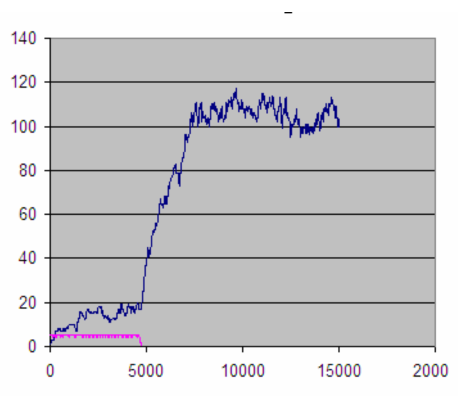
Signal Response of Low-Delay Gate



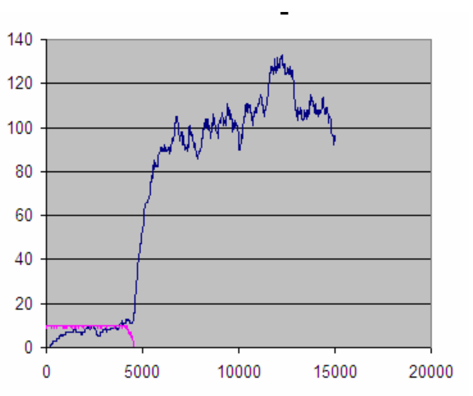
input = 0



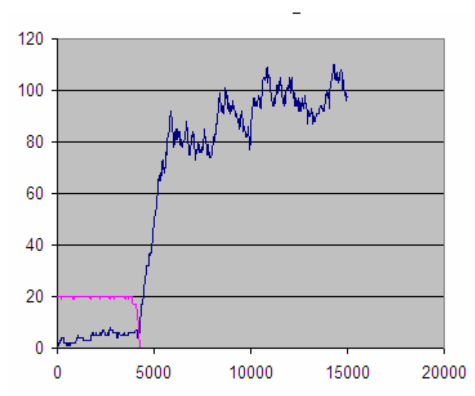
input = 1



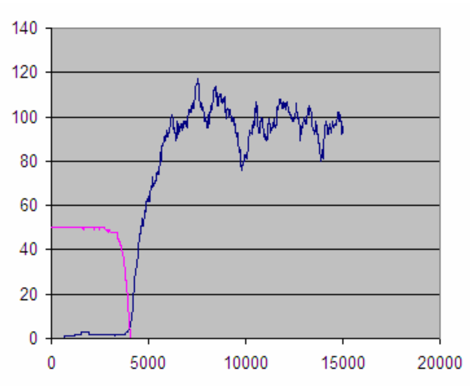
input = 5



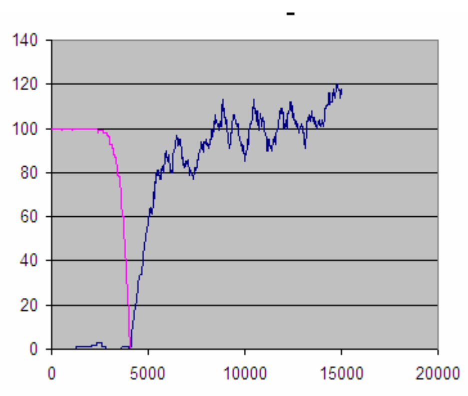
input = 10



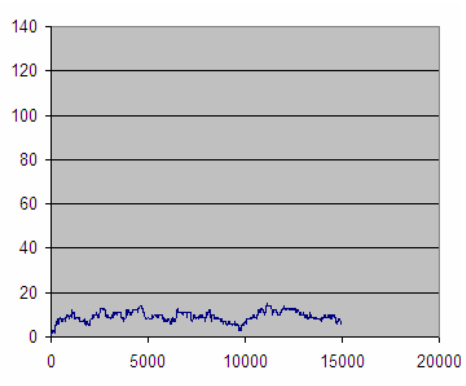
input = 20



input = 50



input = 100



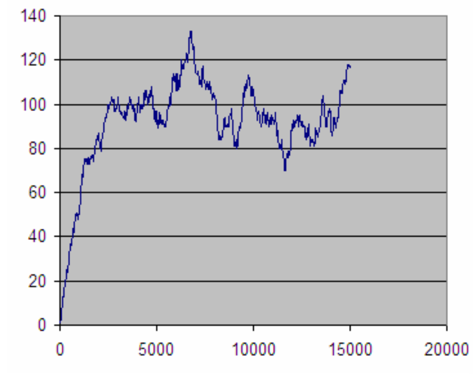
input = output

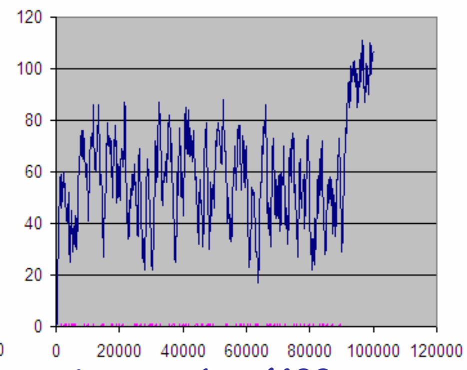
r=1.0, ε=0.1, h=1.0, δ=0.001

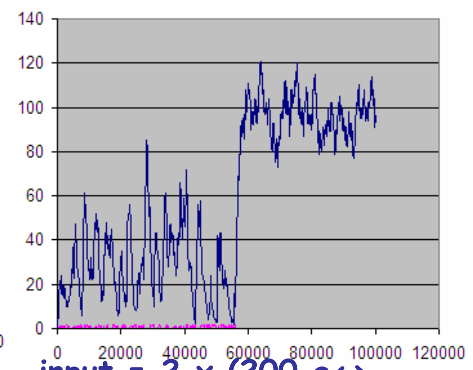
A pretty straightforward antimonotonic gate.

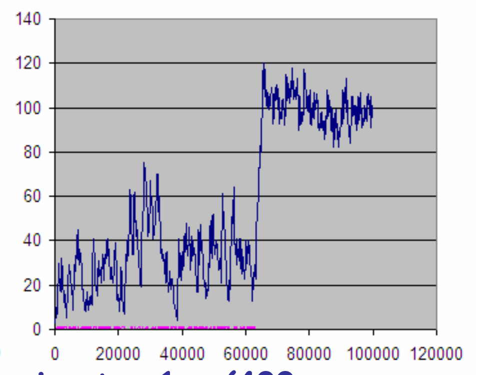
Total# input a<> for each plot (except first and last): 4000.

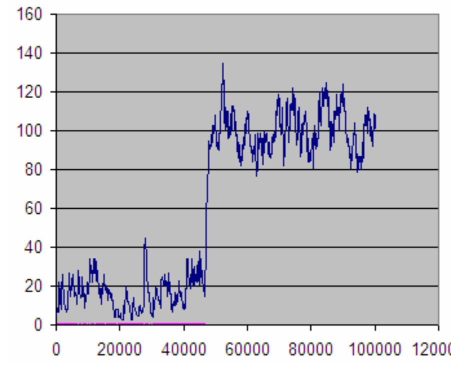
Signal Response of High-Delay Gate

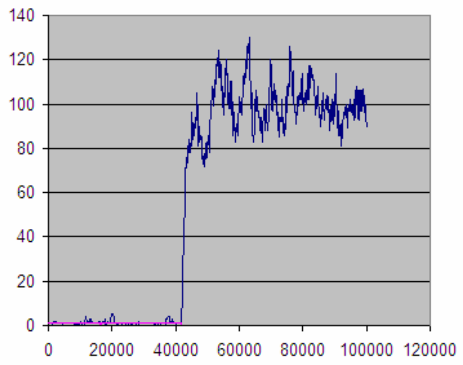

input = 0

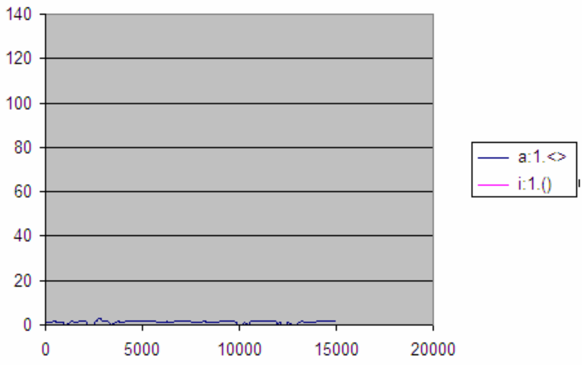

input = 1 x (400 a<>
spaced by delays=0.5*h)


input = 2 x (200 a<>
spaced by delays=0.5*h)


input = 1 x (400 a<>
spaced by delays=h)


input = 1 x (400 a<>
spaced by delays=2*h)


input = 1 x (400
consecutive a<>)



input = output

r=1.0, ε=0.1, h=0.01, δ=0.001

Conclusions

I don't know how the parameters I have been using reflect reality. The point was to show that there exists a reasonably stable (within 1 order of magnitude) set of parameters that gives the "expected" behavior, even for a very simplified model. Therefore there is hope that further refinements will work as well.

Biggest architectural surprise: designing the components so that arbitrary feedback patterns are handled without deadlocks. This does not reflect any common software concurrency paradigm. The closest analogy seems to be with hardware [McAdams&Shapiro].

Biggest engineering surprise: the stabilizing and destabilizing effects of degradation, and the cost of keeping it running on purpose.