Journal of Computer Science & Systems Biology - Open Access

JCSB/Vol.2 September-October 2009

Research Article OPEN ACCESS Freely available online doi:10.4172/jcsb.1000042

Genetic Networks Described in Stochastic Pi Machine (SPiM) Programming Language: Compositional Design

Andrew Kuznetsov*

Freiburg University, Baden-Wuerttemberg GERMANY

Abstract

If biological objects are created by natural selection, why are they composed of discrete modules? What has been the nature of mutations since the Darwinian epoch? This paper presents examples of genetic circuits in terms of stochastic π-calculus; a new mathematical language for nanosystems. The author used a constructor of five elements such as decay, null gate, gene product, and negative and positive gates. These primitives were applied to design genetic switches, oscillators, feedforward and feedback loops, pulse generators, memory elements, and combinatorial logics. The behaviors of those circuits were investigated – functions, such as oscillations or a spontaneous pulse generation were performed simply, flip-flops between stable states occurred in the noisy environment. The modular essence of π-calculus and the following up features of Stochastic Pi Machine (SPiM) programming language allowed us to change the topology of networks that resembled a gene exchange in nature. Other types of mutations were considered as variations in parameters. Perturbations modified system behavior in unpredictable ways that generated diversity for a possible future design by selection of appropriative variants.

Keywords: Genetic network motifs; Modular design; Stochastic pi-calculus; SPiM programming language

Introduction

Biological entities are different from artificial devices because they are developed by an algorithm that Charles Darwin called natural selection. Evolutionary simulations demonstrated that modular structures are rare and less optimal than fully wired counterparts (Poli et al., 2008; Thompson, 1998). That is why the masterpiece of biological networks is an entanglement. However, there are many examples of modular design in evolution; body plans of invertebrates and vertebrates, and modular metabolic networks within bacteria (Kreimer et al., 2008; Parter et al., 2007). Modular structures can spontaneously emerge if the environment changes over time (Kashtan and Alon, 2005). Modularity can also dramatically speed up evolution (Kashtan et al., 2007). On the other hand, modularity is a basis of our ability to divide a problem into parts and to scale up to large systems by a composition of elements.

The motivation of this experiment is a biologically inspired bottom-up design from basic primitives or modules which resemble the natural evolution process of a composition of gene clusters and protein domains (Kuznetsov, 2009a). Artificial genetic networks of increasing complexity were constructed from their components. An attempt to study in a formal way the relation between genotype and phenotype was done and the behavior of genetic circuits of different topologies was investigated. Special attention was given to using an adequate language to describe a large system with interacting components. In general, π-calculus is such a language. Modules were described in the π-calculus formalism because it is subsequently expected to explore a larger system of the modules, like genetic networks. The main point here is a modular nature of π-calculus that opens the door to the combinatorial design on the basis of Stochastic Pi Machine (SPiM) programming language.

Basically, π-calculus is a model of computation for concurrent systems whose configuration may change during the computation in which "everything is a process" and all computation proceeds by communication on channels. The syntax of π-calculus lets us represent parallel processes, synchronous communication between processes through channels, and nondeterminism. A *process* is an abstraction of independent threads. A *channel* is an abstraction of the communication between two processes. Processes interact with each other by sending and receiving *messages* over channels. The content of messages is also channels. As a result of such a communication event, the recipient process may now use the received channel for further communication. This feature, called mobility, allows the network "wiring" to change with interaction. Stochastic π-calculus is an elegant approach to simulating the populations of biological molecules. A system is defined by a flow chart that gives the rates of transitions between states, for example the rate of protein production. The simulator chooses among the possibilities at random simulation. The SPiM is a programming language for designing and simulating computer models of biological processes. The language is based on π-calculus formalism and the simulation algorithm is based on chemical kinetics theory. The language can be used to model large systems incrementally by directly composing simpler models of subsystems.

Some historical facts in the development of π-calculus should be mentioned: the π-calculus was invented in 1992 by Robin Milner with the aim of describing interactive systems. He showed that λ-calculus can be expressed in the π-calculus formalism that demonstrates a computation equivalence of π-calculus and a Turing machine (Milner, 1992; Milner, 1999). The π-calculus has been used to describe many different kinds of concurrent systems from mobile nets to business applications. Aviv Regev and Ehud Shapiro used stochastic π-calculus for a representation and simulation of molecular

***Corresponding author:** Andrew Kuznetsov, Freiburg University, Baden-Wuerttemberg GERMANY, E-mail: andrei_kouznetsov@hotmail.com

Received September 29, 2009; **Accepted** October 29, 2009; **Published** October 29, 2009

Citation: Kuznetsov A (2009) VMD: Genetic Networks Described in Stochastic Pi Machine (SPiM) Programming Language: Compositional Design. J Comput Sci Syst Biol 2: 272-282. doi:10.4172/jcsb.1000042

pathways in computational biology (Priami et al., 2001). Various π-calculus languages with particular formal syntaxes and semantics, as well as simulators for parallel processes were developed; the stochastic π-calculus was originally proposed by Corrado Priami in 1995, while Andrew Phillips and Luca Cardelli developed SPiM language and designed a stochastic simulator (Stochastic Pi-Machine, SPiM) that performed a Gillespie algorithm (Gillespie, 1977). The machine has been formally specified and approved for the stochastic π-calculus (Phillips, 2009b). The SPiM simulator is available for free distribution on different platforms (Linux, MacOS X, Windows). SPiM Player, a graphical user interface to SPiM was written by James Margetson and co-workers in F# language. Molecular systems were described via these tools as collectives of interacting automata in artificial chemistry (Cardelli, 2009). Recently, the SPiM calculus was promoted as a formal visual programming language for biology (Phillips, 2009a).

With this background, the aim of this paper was to investigate abstract gene-regulated networks in terms of basic primitives and their interactions. Following (Blossey et al., 2006), the SpiM simulator based on stochastic π-calculus was used for the modeling of concurrent biochemical processes during the bottom-up design of genetic networks.

The paper is structured as follows. The basic circuit primitives in SPiM calculus are defined in the "method". The section "results" describes the genetic circuits of increasing complexity such as ***neg*** and ***pos*** gates, circuit connections, switches, oscillator motif and repressilator, feedforward and feedback loops, memory elements and bifans. Genetic gates are defined in the SPiM programming language and showed in bold in the text and figures. In addition, the behaviors of gates are described through programming code in the "appendix" and the "supplement". The main contribution of the paper – valid modular design of genetic networks in SPiM language, as well as novel aspects of design are highlighted in the "discussion". For instance, the design job in SPiM programming language looks like an evolution process. Genetic gates with multiple inputs allowed us to develop genetic networks with a high connectivity. A robust memory element consisting of four ***neg*** genetic gates was successfully created. The stochastic noise in SPiM calculus demonstrated a novel phenomena. Some of the data is included in the "supplement".

Method

The Stochastic Pi-Machine Version 0.05 (Phillips, 2007) and the SPiM Player Version 1.13 were used by the author for *in silico* experiments with Microsoft Windows XP operating system and NET Framework 2.0 on a Fujitsu Siemens Computer; Intel Celeron M CPU 520 at 1.60 GHz, 95 MHz, and 448 MB of RAM.

A gap between the abstract mathematical model and the programming code, i.e. between the specification and the program that implements it was eliminated by compression as in earlier studies (Blossey et al., 2006; Blossey et al., 2008). Interactions between biological molecules such as DNA and protein transcription factors were considered as channels. Each element of the network is a genetic gate and defines an input/output relationship corresponding to the binding and synthesis of transcription factors (Blossey et al., 2008). Network elements become autonomous and only the input/output relations determine their wiring. Unary reactions, i.e. protein degradation are followed by a stochastic delay. Binary reactions are represented as a channel with an input (?) and output (!). A protein reactant is defined as an input or output on a given channel. Extra notations are used in the paper such as the constitutive expression rate $\boldsymbol{\varepsilon}$, protein degradation $\boldsymbol{\delta}$, gene unblocking $\boldsymbol{\eta}$, and **r** for the binding of the transcription factor. For instance, a transcription factor can decay in the reaction $\boldsymbol{\delta}$, a blocked gene can be unblocked by the reaction $\boldsymbol{\eta}$. A gene can also produce a new protein independently in the reaction $\boldsymbol{\varepsilon}$, where a vertical bar (|) represents parallel execution (see definitions **1-5** below). An initial population of proteins is empty; they are expressed constitutively and stochastically by genetic gates.

A stochastic nature of molecular interactions on a nanoscale was described according to (Blossey et al., 2006). For example, $\boldsymbol{\tau}_{\varepsilon}$ is a stochastic delay where $\boldsymbol{\tau}$ is a symbol indicating event delay, and $\boldsymbol{\varepsilon}$ is the stochastic reaction constant which gives the probability per unit time that the delay action will occur (Gillespie, 1977). When an action with rate $\boldsymbol{\varepsilon}$ is enabled, the probability that it will happen in the period of time $\boldsymbol{t}$ is $\mathbf{P}(\boldsymbol{t}) = \mathbf{1}\text{-}\mathbf{e}^{-\varepsilon t}$. This distribution satisfies Markov's property of stochastic dynamics.

The following primitives in SPiM calculus were used to build networks with different topologies:

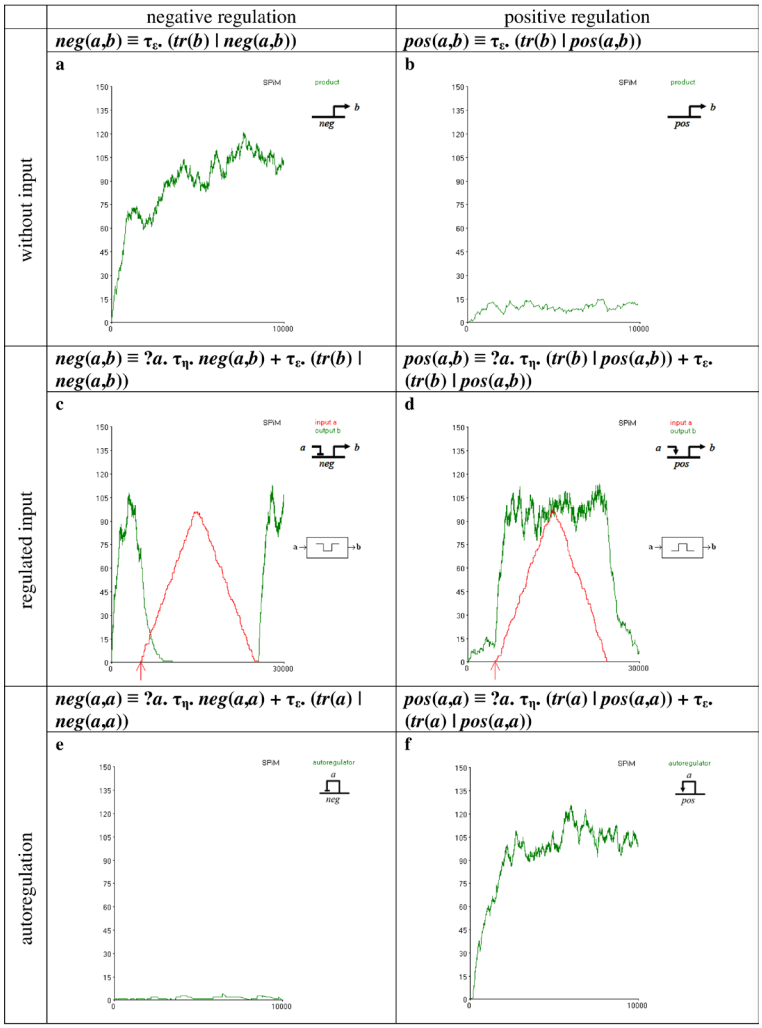
1) decay or degradation of a transcription factor $\boldsymbol{tr(b)} \equiv \tau_{\delta}$,
2) null gate or constitutive transcription $\boldsymbol{null(b)} \equiv \boldsymbol{\tau_{\varepsilon}.\ (tr(b) \mid null(b))}$,
3) gene product or protein transcription factor $\boldsymbol{tr(b)} \equiv \boldsymbol{!b.\ tr(b) + \tau_{\delta}}$,
4) ***neg*** gate or negative regulation $\boldsymbol{neg(a,b)} \equiv \boldsymbol{?a.\ \tau_{\eta}.\ neg(a,b) + \tau_{\varepsilon}.\ (tr(b) \mid neg(a,b))}$,
5) ***pos*** gate or positive regulation $\boldsymbol{pos(a,b)} \equiv \boldsymbol{?a.\ \tau_{\eta}.\ (tr(b) \mid pos(a,b)) + \tau_{\varepsilon}.\ (tr(b) \mid pos(a,b))}$.

Genetic networks with an increasing number of nodes and connectivity were investigated with a nominal set of parameters: the protein binding $\mathbf{r} = 1.0$ and protein decay $\boldsymbol{\delta} = 0.001$ for any gate, the constitutive transcription $\boldsymbol{\varepsilon}_n = 0.1$ and repression $\boldsymbol{\eta}_n = 0.01$ for a negative gate, and the constitutive transcription $\boldsymbol{\varepsilon}_p = 0.01$ and activation $\boldsymbol{\eta}_p = 0.1$ for a positive gate. To exploit a parameter space, I gradually varied the rates **r, δ, ε** and **η** in the experiments. Algorithms were mapped to executable program codes for SPiM (see **Appendix** and **Supplement 4**).

Results

Behavior of basic gates

The basic ***neg*** and ***pos*** gates were represented previously by Blossey and co-workers (Blossey et al., 2006). In the case of a negative gate (4) without input shown in Figure 1a, the gene can express a protein by performing first a stochastic delay at the rate $\boldsymbol{\varepsilon}_n = 0.1$ and then executing a new protein in parallel to the gene itself. Alternatively, in the presence of input (Figure 1c), it can be blocked by an input on its promoter region ***a*** and then be unblocked during a stochastic delay at the rate $\boldsymbol{\eta}_n = 0.01$ to activate the gene. In contrast, the modified positive



Standard parameters	negative gate	positive gate
protein binding, $\mathbf{r} = 1.0$	constitutive transcription, $\varepsilon_n = 0.1$	constitutive transcription, $\varepsilon_p = 0.01$
protein decay, $\delta = 0.001$	repression, $\eta_n = 0.01$	activation, $\eta_p = 0.1$

Figure1: Basic genetic gates described in stochastic π-calculus.

genetic gate (5) depicted in Figure 1b, can follow in stochastic delay at the low rate $\varepsilon_p = 0.01$ to constitutively express a new protein in parallel to the gene or it can carry out an input on its promoter region ***a*** and then activate transcription at the high rate $\eta_p = 0.1$ (Figure 1d).

Simulations were started in the absence of proteins by doing a constitutive transcription. The number of protein molecules initially increased and finally leveled off at equilibrium between the production and degradation depending on the parameters **ε** and **δ**. The constitutive expression and the output were higher for the negative regulator than for the positive one (Figure 1a, b).

To observe a response of genetic elements, an input allowed linear increases from 0 to 100 individual molecules, then decreased linearly to 0. The input *a*-molecules were injected into the system at a certain time to shape the curve *b*. As a result of the reaction, the negative gate behaved like an inverter (Figure 1c) whereas the positive gate increased the output signal about almost 10 times (Figure 1d). For each plot, the abscissa indicates the time of simulation and the ordinate is the number of molecules.

Apparently, the simplest circuit is the single gate interacting with itself in a feedback loop. In this case, a promoter region ***a*** of the gene is parameterized together with the transcribed protein ***tr*(*a*)**. The transcription factor can repeatedly make an output acting on the promoter region ***a***, or it can decay at the rate **δ** according to expression (3). If output was wired to input, then the negative and positive gates showed an autorepression and an amplification respectively (Figure 1e, f). The ***neg*(*a*,*a*)** gate stabilized at an appropriative low level of ***tr*(*a*)**. In contrast, a magnitude of ***tr*(*a*)** for the ***pos*(*a*,*a*)** gate was about 100 times higher in my experiments.

Connection of components in networks

Once basic elements are defined, genetics circuits can be assembled by providing interaction channels connecting various gates. For example, ***pos*(*a*,*b*) | *pos*(*b*,*c*)** means the ***pos*(*a*,*b*)** process will offer output ***!b*** through ***tr*(*b*)** and the ***pos*(*b*,*c*)** will ask for input ***?b***. Hence, the shared channel ***b*** can result in interaction between the two processes. The same principle can be extended to a chain of gates (Figure 2a).

The simulation took place with a constitutive transcription. Channels were declared separately. In the absence of stimulus, the first ***pos*** gate produced a low output *b* which was enough to activate the next element. The outputs of the ele-

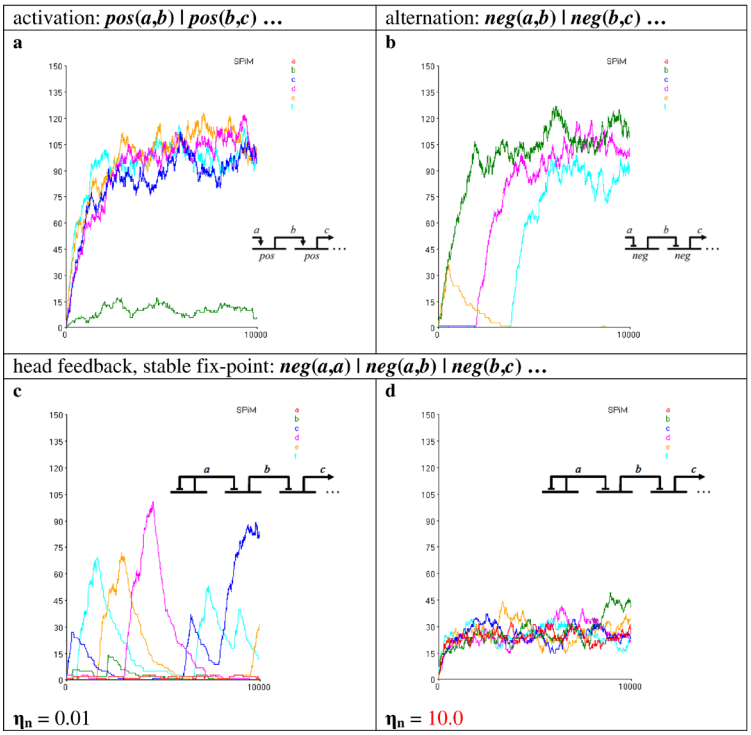


Figure 2: Composition of functional genetic networks

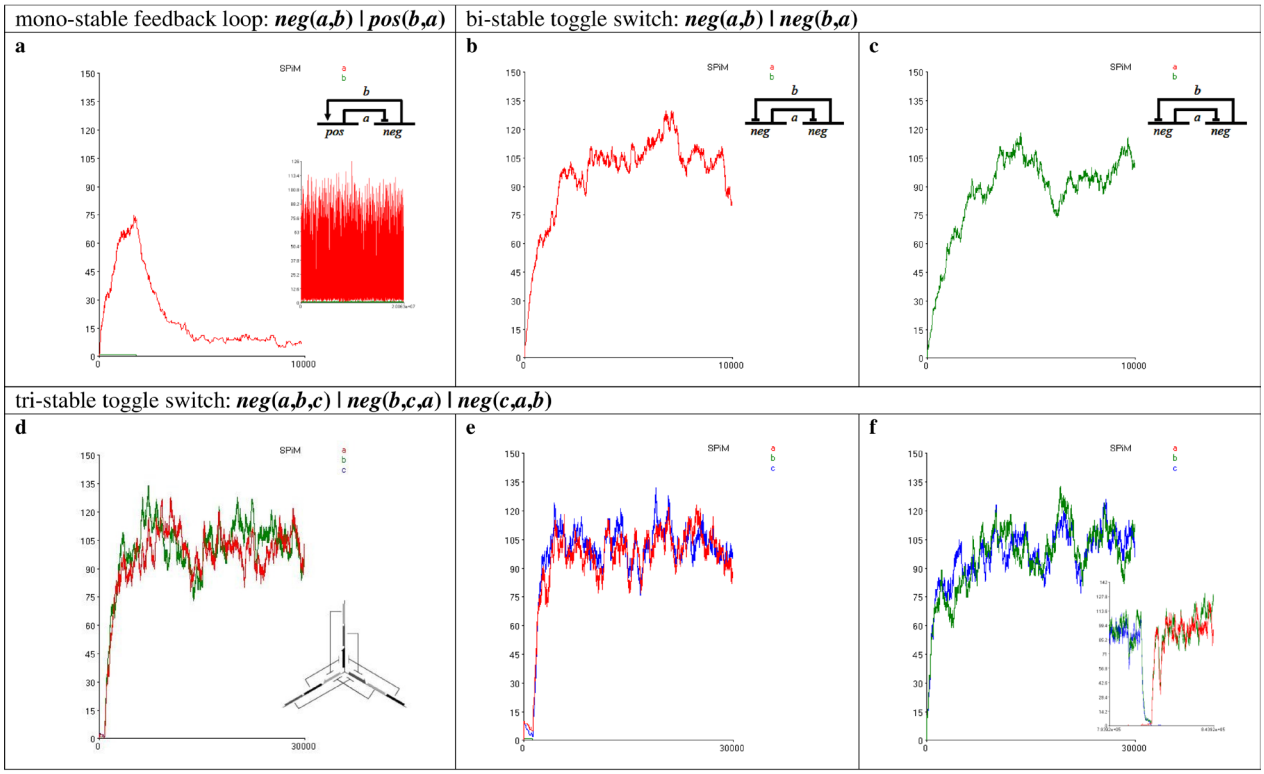


Figure 3: Genetic switches.

ments that followed increased very rapidly. In contrast, a sequence of ***neg*** gaits demonstrated an interchange of outputs. In the absence of an initial input *a*, the odd gates showed high output values *b*, *d*, *f*. At the same time, outputs of the even gates *c*, *e* stayed at a low level (Figure 2b). A signal moved through the chain of ***neg*** gates with a delay depending on the parameter η_n. The smaller the value of η_n, the longer the time for gene activation and the slower the propagation of the activation wave (data not shown).

The same circuit including a negative feedback on the head gate showed a rather altered behavior. The performance was chaotic at parameter $\eta_n = 0.01$. Gene activities fluctuated significantly by time and amplitude (Figure 2c). When the repression delay was increased to $\eta_n = 10.0$, the system stabilized itself at a fix-point (Figure 2d). These results comply with Blossey and co-authors (Blossey et al., 2006).

Switches

The combination ***neg(a,b) | pos(b,a)*** is a mono-stable feed-

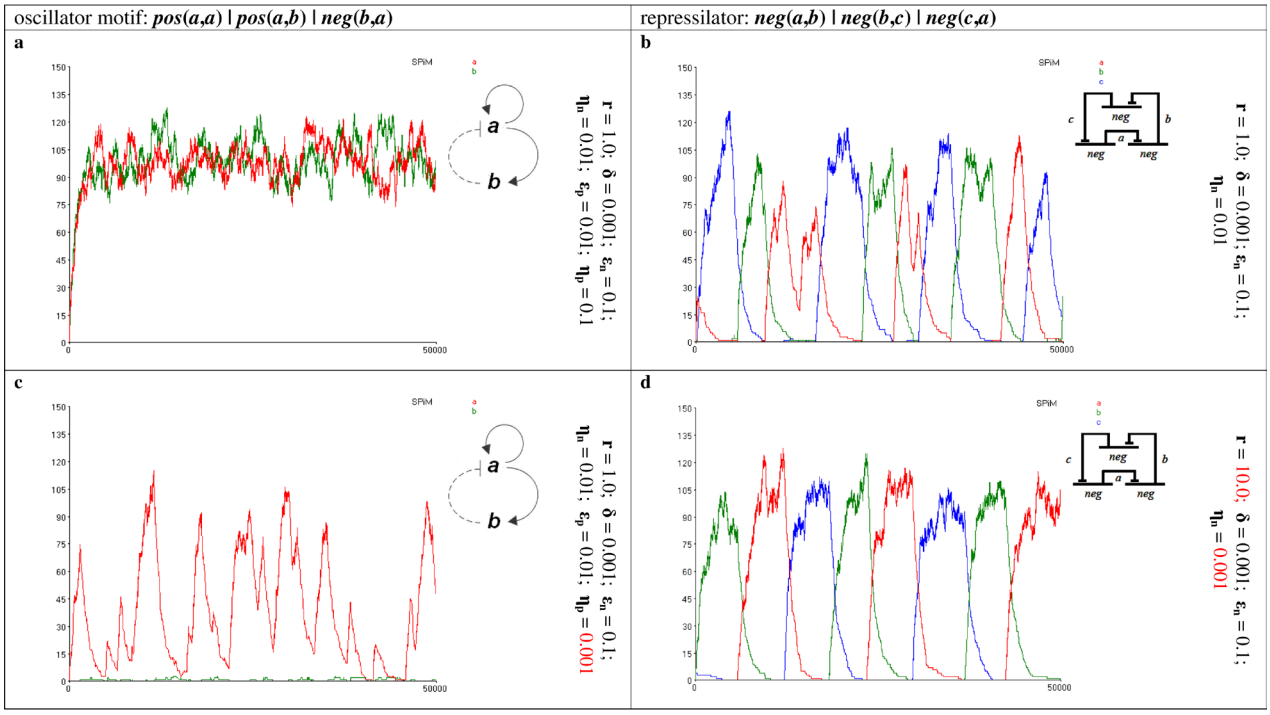


Figure 4: Oscillator motif and repressilator.

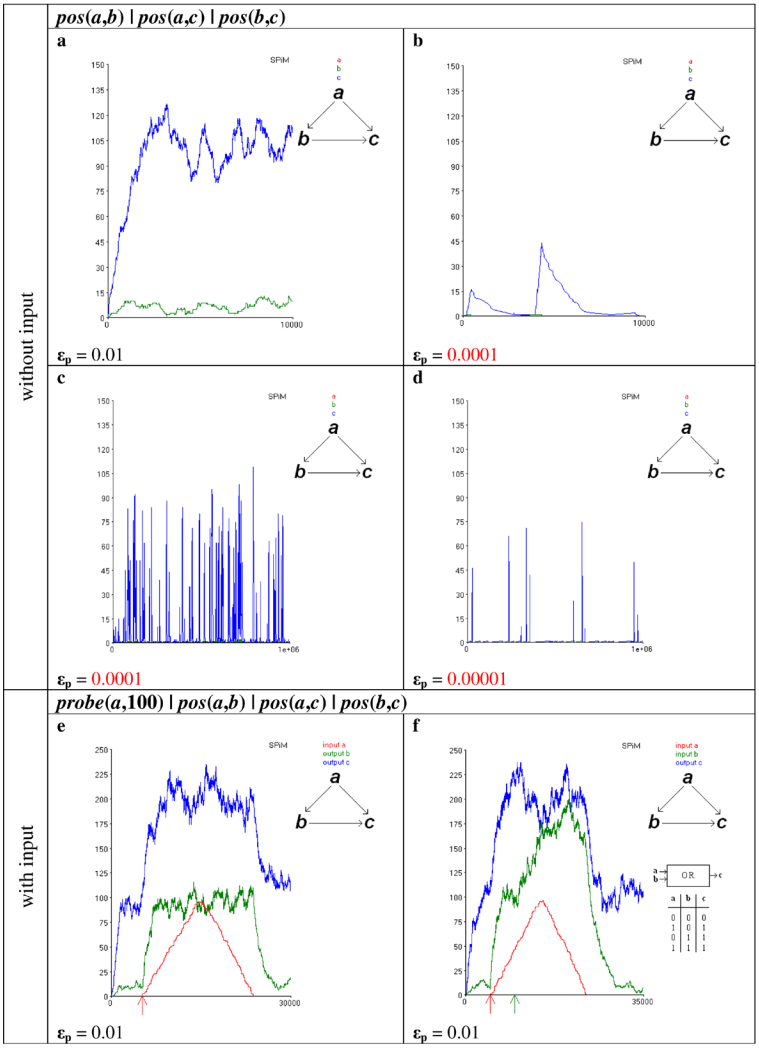


Figure 5: Coherent feedforward loop (C1-FFL).

back loop. In contrast to ***neg*(*a*,*a*)**, it has two outputs *a* and *b*. In the absence of stimuli, the system can stochastically start with an occurrence of protein *a* before stabilization (Figure 3a). A long-time simulation revealed spontaneous fluctuations in the expression of this protein, up to 126 molecules in a spike (Figure 3a, insertion).

The composition ***neg*(*a*,*b*) | *neg*(*b*,*a*)** is a bi-stable toggle switch, which can start up in one state or another and as a rule stay there (Figure 3b, c).

More interesting is a tri-stable toggle switch ***neg*(*a*,*b*,*c*) | *neg*(*b*,*c*,*a*) | *neg*(*c*,*a*,*b*)** that can produce three alternative combinations of proteins, such as (*a*,*b*), (*b*,*c*) and (*c*,*d*). Each gate has two promoter regions and one output. They are wired to each other as shown in Figure 3d. Occasionally, the system did flip-flops from one state to the other at the conventional set of parameters (Figure 3f, insertion). When a repression by protein binding **r** was increased to the value 10.0 instead of 1.0, the toggle switch showed stable behavior for a long time. Thus, the tri-stable toggle switch demonstrates a possibility to build nontrivial genetic networks with high connectivity.

Oscillator motif and repressilator

An oscillator motif includes a negative feedback loop and a positive self-activation loop. It can be written in SPiM language such as ***pos*(*a*,*a*) | *pos*(*a*,*b*) | *neg*(*b*,*a*)**. Preliminary experiments with standard values of parameters did not reach oscillations. Proteins fluctuated due to inherent stochastic noise in the system (Figure 4a). When the rate of positive gene unblocking was decreased to $\eta_p = 0.001$, irregular oscillations of protein *a* were induced (Figure 4c).

A repressilator consists of three ***neg*** gates that mutually repress each other: ***neg*(*a*,*b*) | *neg*(*b*,*c*) | *neg*(*c*,*a*)**. Simulation of the repressilator at nominal parameters resulted in an irregular duration of protein cycles (Figure 4b). Since dynamics of the network depends on relative rates of the parameters **r, δ, ε** and **η** (Blossey et al., 2008), it was possible to fix the value of one parameter ($\varepsilon_n = 0.1$) in order to study the effects of the others. The decreased rate of gene unblocking $\eta_p = 0.001$ and the increased protein binding $r = 10.0$ allowed an improvement of the regularity of oscillations. Populations of proteins stabilized near 100 molecules in each cycle with the duration of impulses being about 625 units (Figure 4d).

Feedforward loops

A feedforward loop (FFL) is a circuit from three genes that express two transcription factors one of which regulates the other and together they regulate a target gene. The FFL has eight possible subgraphs, however, only two of them, namely the C1 coherent feedforward loop (C1-FFL) and the I1 incoherent feedforward loop (I1-FFL) are highly abundant in transcription networks of *E.coli* and yeast (Alon, 2007). These motifs were denoted as follows: C1-FFL – ***pos*(*a*,*b*) | *pos*(*a*,*c*) | *pos*(*b*,*c*)** and I1-FFL – ***pos*(*a*,*b*) | *pos*(*a*,*c*) | *neg*(*b*,*c*)**. Variations in a constitutive expression of ***pos*** gates (ε_p) were critical for both feedforward loops.

C1-FFL output at nominal parameters and an empty start condition is depicted in Figure 5a. Curves demonstrate a correlation between amounts of *b* and *c* proteins, depending on ***pos*(*b*,*c*)** gate activity. The system stabilized at low records for protein *b* and a high quantity for protein *c* in agreement with the unitary regulation for *b* and double positive regulation for *c* proteins. When the value of ε_p decreased, the amounts of both proteins dropped considerably. Spontaneous impulses of *c* protein expression were observed at the very small $\varepsilon_p = 0.0001$, indicating a fast front and an exponential backside and amplitude up to 100 molecules (Figure 5b, c). When ε_p went down to 0.00001, the time between impulses increased and they became far less frequent, and the system mostly slept (Figure 5d). This behavior is determined by very rare constitutive transcription events at ***pos*(*a*,*c*)** and ***pos*(*b*,*c*)** genes because of the enormous delay ε_p. It may be that only a successful combination of activity for ***pos*(*a*,*c*)** and ***pos*(*b*,*c*)** gates with dependency on ***pos*(*a*,*b*)** gate is able to activate *c* output under the conventional conditions of relatively hard protein degradation $\delta = 0.001$.

As usual, the standard set of parameters was used in experiments with input. Proteins *a* and *b* were injected at 500 and at 1000 time intervals of simulation according to a triangle-shape function. In the case of single input *a*, Figure 5e shows a sharp response of *b* and *c* proteins with saturation at 100 and 200 protein molecules respectively. A double injection did not change the shape of *c* output even during the simultaneous presence of both proteins *a* and *b* (Figure 5f).

I1-FFL output at nominal parameters was depressed via an antagonism between ***pos*(*a*,*c*)** and ***neg*(*b*,*c*)** gates (Figure 6b). When the value ε_p increased, it meant an augmentation of constitutive transcription for the ***pos*** gates and production of both proteins *b* and *c* improved significantly (Figure 6a). In this situation, the role of the ***neg*(*b*,*c*)** gate was insufficient. On the other hand, when the constitutive transcription for ***pos*** gates decreased to $\varepsilon_p = 0.001$, then the situation changed and a fine balance between ***pos*(*a*,*c*)** and ***neg*(*b*,*c*)** gates played a major role. As a rule, the productivity of ***neg*(*b*,*c*)** gate was enough to suppress *c* protein for a time. Warped flops in *c* output were observed as a result of chaotic ***neg*(*b*,*c*)** regulation activity (Figure 6c). When a value of transcription for *pos* gates decreased to $\varepsilon_p = 0.00001$, the activation of ***neg*(*b*,*c*)** gate by ***pos*(*a*,*c*)** was very low and the system demonstrated a stable production of *c* protein once again (Figure 6d). Under these conditions, the ***neg*(*b*,*c*)** gene was able to inhibit *c* output only sporadically (Figure 6d, insertion).

Subsequent simulations with inputs were performed at nominal parameters as before. An injection of protein *a* in the system provoked a fast coherent response for both *b* and *c* proteins up to 100 molecules (Figure 6e). Additional injection of protein *b* did not affect *c* output (Figure 6f). When injections were finished, the system retuned into an initial state.

Obtained data showed that C1-FFL circuit behaved like a logical OR operator, whereas I1-FFL gate demonstrated TRUE-A logics under the standard conditions: $r = 1.0$, $\delta = 0.001$, $\varepsilon_n = 0.1$, $\eta_n = 0.01$, $\varepsilon_p = 0.01$ and $\eta_p = 0.1$.

Composition of feedforward and feedback loops (FFBL)

Coherent and incoherent FFBL were denoted as the ***pos*(*a*,*b*) | *neg*(*b*,*a*) | *pos*(*a*,*c*) | *neg*(*b*,*c*)** and the ***pos*(*a*,*b*) | *neg*(*b*,*a*) | *neg*(*a*,*c*) | *pos*(*b*,*c*)** accordingly. They operated at a nominal regime like coherent pulse generator and decoherent pulse generator (Figure 7). Nevertheless, variations in parameters lead

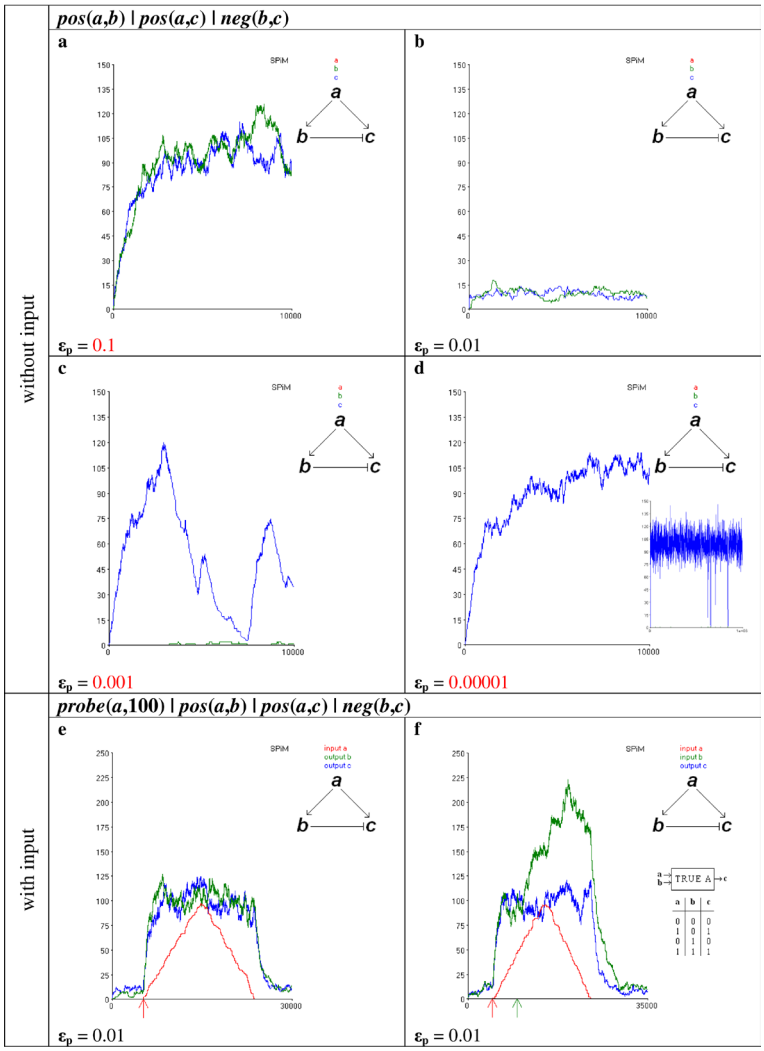


Figure 6: Incoherent feedforward loop (I1-FFL).

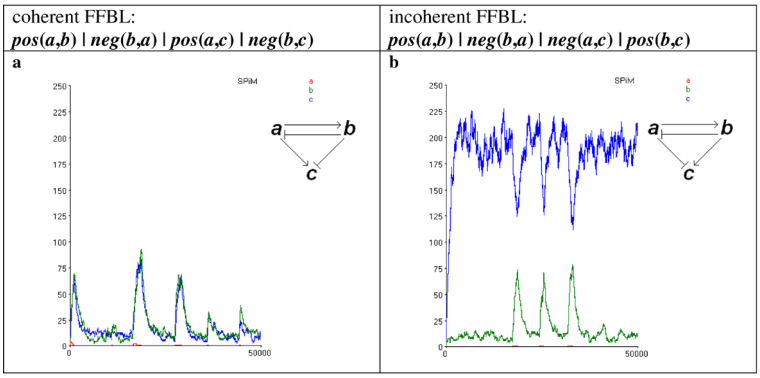


Figure 7: Coherent and incoherent compositional feedforward and feedback loops - stochastic pulse generators.

to extremely diverse behaviors in experiments. The rates of parameters ε_n, η_n, ε_p and η_p were changed gradually from 0.1 to 0.0001. Both FFBLs were investigated in detail during short and long simulations (up to $5 \cdot 10^4$ and 10^6 time intervals). Results of the experiments are in Supplements 1 and 2.

I discovered that the coherent FFBL circuit can be in four different states corresponding to particular patterns of protein expression: I – low basal production of *b* and *c* proteins, II – intensive stable production of *b* and *c*, III – spontaneous synchronous outputs of *a* and *c*, and IV – exhaustive expression of *a* and *c* proteins with gaps. These essential states were significantly influenced by stochastic fluctuations. Nevertheless, I did not observe any transition from one state to another when the parameters were fixed. A system with a standard set of parameters was typically in state I which demonstrated an equilibrium level of *b* and *c* proteins near 10 molecules and synchronous pulses of expression up to 100 molecules (S1: pictures 1, 6, 10, 13). When the value of parameter ε_n was progressively decreased from 0.1 to 0.0001, then the frequency and amplitude of these pulses fell; the system reached a still state with low expression (S1: 1-4). Hence, a silencing of the constitutive expression for negative regulators reduced the

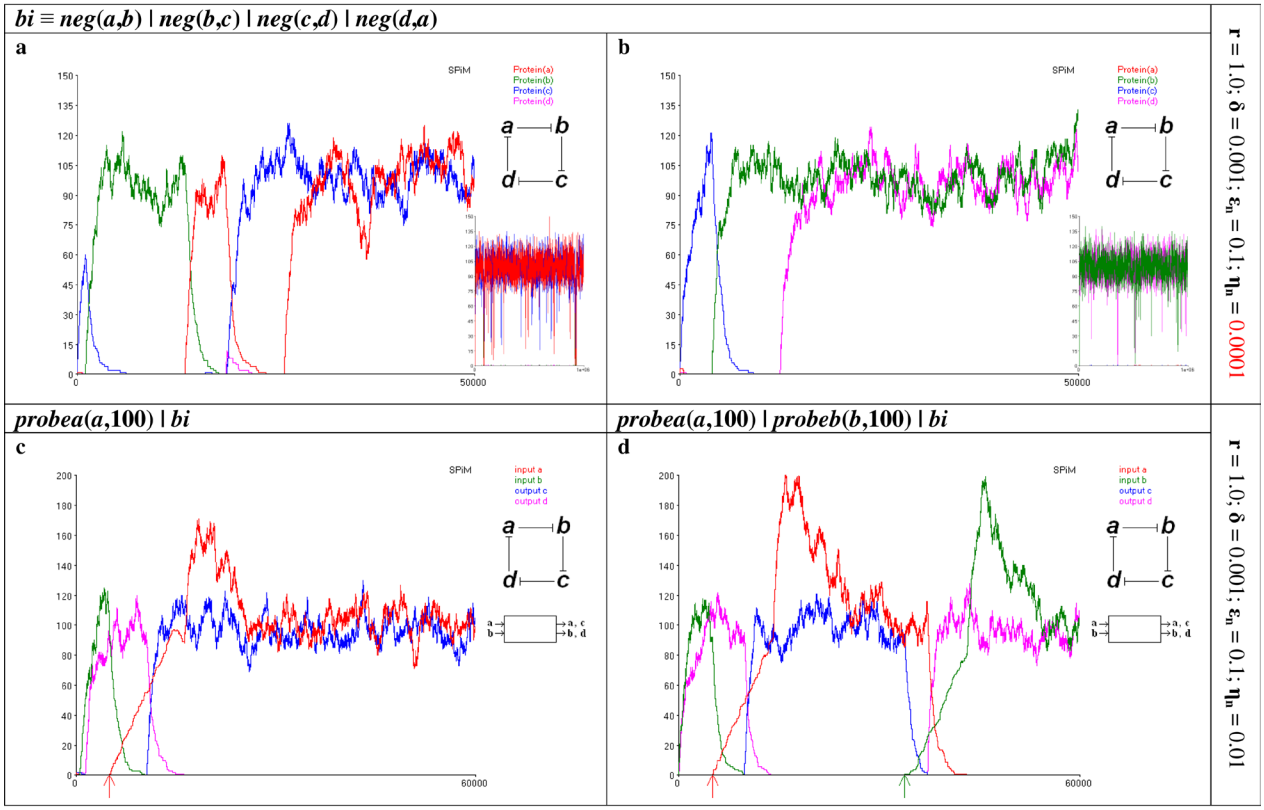


Figure 8: Bi-stability and a memory effect within 4-node feedback loop.

coherent FFBL to low activity at the standard rates of other parameters.

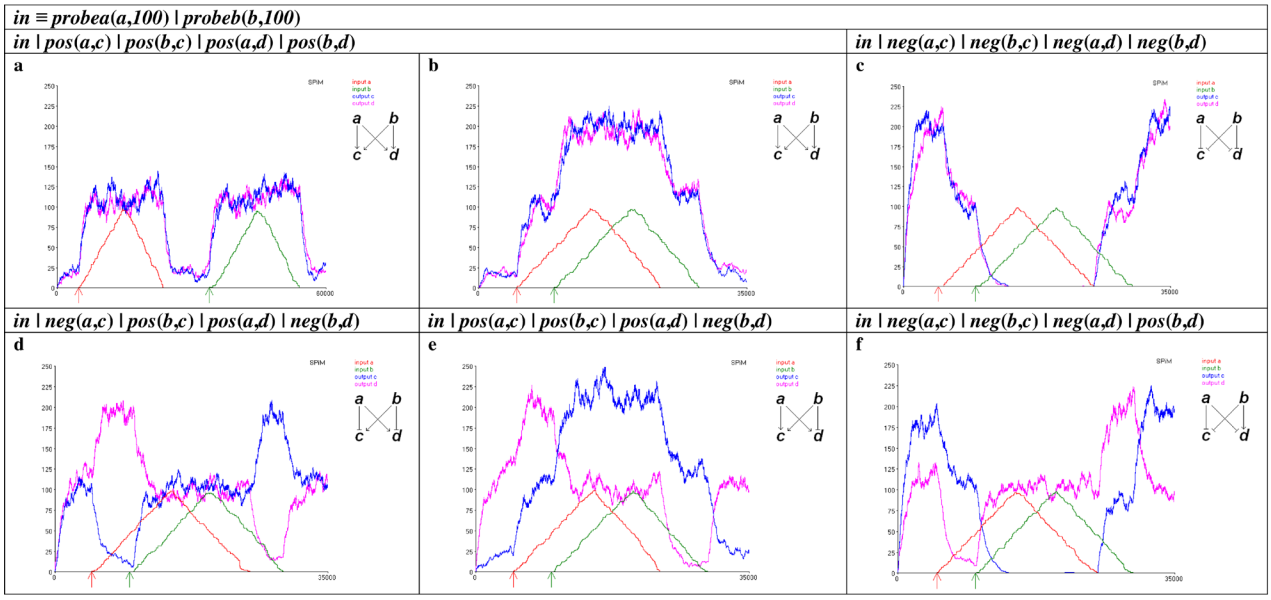
When the value of η_n was altered from 0.01 to 0.1, then the system generated more pulses with the amplitude up to 100 molecules (S1: 5). A shift of η_n from 0.01 to 0.0001 led to a low protein expression of about 10 molecules (S1: 8) as in the previous still state. Small values of parameters ε_n and η_n determined the same output effects mediated by ***neg*** gates (S1: 4 and 8). Obviously, a rare unblocking ***neg*** regulation has a similar effect as the low constitutive expression.

When I investigated the properties of ***pos*** gates, the value of ε_p increased to 0.1 and an expression of *b* and *c* proteins increased to 100 molecules; it is just a state defined as II (S1: 9). A production of those proteins significantly improved as a result of a more effective ***pos*(*a*,*b*)** and ***pos*(*a*,*c*)** constitutive expression. In contrast, when the ε_p decreased, the frequency of synchronous positive pulses increased (S1: 11, 12).

Activities of the coherent FFBL were more surprising when the parameter η_p changed. The system occurred in an unruffled state I at value $\eta_p = 0.01$ with a low level of expression for *b* and *c* proteins (S1: 14) exactly like the small rates of ε_n and $\eta_n = 0.0001$ (S1: 4, 8). However, the system stayed in a new state III at $\eta_p = 0.001$, producing proteins *a* and *c* synchronously instead of *b* and *c*. From time to time, the circuit generated complex pulse trains of *a* and *c* proteins with an amplitude of up to 100 molecules (S1: 15). A poor basal production of protein *b* at an average of 10 molecules interchanged with these islands for an intensive expression of *a* and *c* proteins. When the η_p had a rate 0.0001, then the system occupied the next state IV with a rich synchronous output of *a* and *c* proteins and negative pulses to a low content (S1: 16). Thus, the progressive decreasing of probability of activation for positive gates ***pos*(*a*,*b*)** and ***pos*(*a*,*c*)** led firstly to the stabilization of *b* and *c* expression at a low level (S1: 14), but later provoked an intensive production of protein *a* instead of *b* through a feedback loop. In the case of infrequent activations of the ***pos*** gates, the system rearranged itself from state I to states III and IV (S1: 15, 16).

An incoherent FFBL circuit showed even more sophisticated outputs. Sometimes the system demonstrated unstable dynamic behavior. I found this circuit in the following states: I – decoherent small amounts of *b* and large amounts of *c* proteins, II – independent low level of *b* protein and intermediate level of *c* protein, III – flip-flops between *a* and *c* production, and IV – asynchronous low *c* and intermediate *a* proteins expression. In addition, I introduced the next states: Ia – for independent low *b* with intermediate and high *c* levels, as well as state IIa – for intermediate *b* and high *c* proteins expression. The low level of any protein expression accorded to about 10 molecules, the intermediate level to near 100 molecules, and the high level averaged 200 molecules. The circuit remained in state I at the conventional values of the parameters. This system generated synchronous pulses with positive and negative amplitudes for *b* and *c* proteins up and down to 100 molecules accordingly (S2: 1). When the rate of ε_n was shifted from 0.1 to 0.01-0.0001, the system followed in state II. In this case, the constitutive expression of ***neg*(*b*,*a*)** and ***neg*(*a*,*c*)** genes decreased. A negative control of protein *a* dropped, hence protein *a* suppressed *c*. The output of *c* decreased from 200 to 100 molecules. In addition, the frequency and amplitude of *b* pulses fell (S2: 2-4).

When I increased the value of η_n up to 0.1, the frequency of *b* and *c* pulses rose as a result of frequent unblocking actions for negative gates (S2: 5). When the value of η_n decreased to



Standard parameters	negative gate	positive gate
protein binding, $\mathbf{r} = 1.0$	constitutive transcription, $\varepsilon_n = 0.1$	constitutive transcription, $\varepsilon_p = 0.01$
protein decay, $\delta = 0.001$	repression, $\eta_n = 0.01$	activation, $\eta_p = 0.1$

Figure 9: Bifan – time-amplitude modulation with different patterns of expression.

0.0001, then I observed an unusual effect which I called state Ia. Sometimes the system dropped to an intermediate level of *c* expression for a short time and after returned to the high *c* production typical for state I (S2: 8). I believe that infrequent unblocking events for negative gates lead the system to a visibly unstable intermediate level of *c* protein expression.

In the following experiment, I increased ε_p to 0.1 that gave a new state IIa with intermediate *b* and high *c* expression (S2: 9). This change of parameters improved the constitutive expression of ***pos***(*a*,*b*) and ***pos***(*b*,*c*) genes leading to a superproduction of *b* and *c* proteins. I identified an incoherent FFBL circuit in the basic state I at $\varepsilon_p = 0.01$, 0.001 and 0.0001 even though the frequency of pulses progressively increased (S2: 10-12).

Finally, when the value of parameter η_p decreased to 0.01, the system woke up and changed from state I to state II (S2: 13, 14). When η_p decreased, the behavior transformation was more dramatic. The system turned on to state IV at $\eta_p = 0.0001$. An expression of protein *a* appeared at an intermediate level up to 100 molecules and the expression of protein *c* decreased drastically to about 10 molecules (S2: 16) which I had not observed before. In addition, I discovered very interesting reactions at $\eta_p = 0.001$: the system switched from time to time from state II to state IV producing intricate pulse trains of the protein *a* or *c* (S2: 15). I defined this state as III based on the instability and sufficient role of specific η_p activation within the feedback loop ***pos***(*a*,*b*), ***neg***(*b*,*a*).

In general, both coherent and incoherent FFBL circuits produce coherent and decoherent pulses, being in state I at a nominal set of parameters. In addition, they behave as stochastic pulse generators in the instable state III at the particular low rate $\eta_p = 0.001$.

Bi-stability and memory effects

Achievements in previous sections were used to design a genetic memory element. The closed chain ***neg***(*a*,*b*) | ***neg***(*b*,*c*) | ***neg***(*c*,*d*) | ***neg***(*d*,*a*) demonstrating bi-stable characteristics was investigated in detail. The system arbitrarily started from expression (*a*,*c*) or (*b*,*d*) proteins. I investigated the circuit for different values of parameter η_p but did not detect spontaneous transitions between states even at the extremely low $\eta_p = 0.00001$. Flip-flops were detected at the constitutive transcription $\varepsilon_n = 0.01$. Effective protein binding $\mathbf{r} > 1.0$ improved stability, however intensive protein decay $\delta > 0.0001$ destabilized the system (data not shown). Fortunately, the circuit demonstrated stable behavior at a standard range of parameters. After it dropped to an arbitrary state, the system survived for a long time (Figure 8a, b). However, the circuit was sensitive to external inputs. For example, when the system is in state *bd*, then a programmable input *a* can change the state to a new state *ac*, i.e. the production of *b* and *d* proteins can be changed to *a* and *c* proteins by input *a* (Figure 8c). More of them, if input *a* no longer exists, nevertheless, the system stayed at the state *ac* and did not turn back until the specific input *b* changed the system's state again (Figure 8d). The program code for this experiment is in the Appendix.

Bifan

Bifan is a simplest element of combinatorial logics based on multiple inputs. I investigated bifans with variable structures under the control of diverse inputs. These circuits represented a stable and relatively poor behavior at various parameters. Bifan comprising only ***pos*** gates performed synchronous outputs, when the input signals were applied separately (Figure 9a). Over time, when the inputs overlapped each other, the ***pos*** bifan worked as an adder (Figure 9b). Bifan consisting exclusively of ***neg*** gates behaved as an inverter or a subtractor depending on the time interval between inputs (Figure 9c). Symmetric bifan with two ***pos*** gates and two ***neg*** gates generated symmetric outputs (Figure 9d). Opposite, asymmetric bifans with three ***pos*** gates and one ***neg*** gate (Figure 9e) or

with one ***pos*** gate and three ***neg*** gates (Figure 9f) performed more complex jobs. These patterns were repeatable, recognizable, and strongly dependent on time and a combination of inputs even in the stochastic environment.

Discussion

I wondered how Kaufmann's networks could be combined with agent-based modeling to describe interactions between objects in terms of rules. One idea was to avoid a combinatorial explosion. The answer was found earlier by Robin Milner, Joachim Parrow, and David Walker who introduced π-calculus which are inherently modular. This concurrent language describes a dynamical network in terms of computable elements. Andrew Phillips and colleagues developed SPiM calculus that is in fact a language for biological systems. For example, gene is defined as a gate with corresponding inputs and outputs. Genetic networks are composed from simple computational elements. Modeling with Stochastic Pi-Machine is like programming, where program parts correspond to autonomous genetic elements. The design of a system from essential elements looks like a process of evolution. The "…compositional evolution is inherently scalable and leaves gradual evolution stuck at the starting blocks" (Watson, 2006). Therefore, the composition nature of π-calculus could allow a combinatorial design and perhaps genetic programming in the field.

A genetic constructor of five elements was used in this work; decay, null gate, gene product, negative and positive gates. Actually, the two last primitives are composed from the three first ones and only the gene product, as well as ***neg*** and ***pos*** gates were used to build more complex engines, such as genetic switches, oscillators, feedforward and feedback loops, pulse generators, memory elements and combinatorial logics based on multiple inputs – the essential blocks of large information devices and networks. This flexible toolkit allowed me to investigate genetic circuits of different topologies and complexities. An important part of the simulations was stochastic noise which is an internal property of biological systems that have a small number of molecules in a cell. The SPiM computing environment showed that some functions such as oscillations or stochastic pulses could be performed easily. For example, the SPiM Player used in the experiments is based on the Gillespite stochastic algorithm that enables accurate computer simulations. Unfortunately, the algorithm is computationally too expensive for models with a large number of channels.

From another perspective, motifs in large-scale gene regulation networks were regarded like 'biological atoms' (Alon, 2007). They are negative and positive auto-regulators, coherent and incoherent feedforward loops, single-input module, multi-output feedforward loop, bifan and dense overlapping regulators. These network motifs discovered in nature by systems biology could possibly be used as building blocks for synthetic biology (Kuznetsov, 2009b).

In this paper, artificial and natural networks were investigated. At the beginning, I repeated the original models in SPiM calculus developed by (Blossey et al., 2006), such as inverter (Figure 1c), a chain of ***neg*** gates and the head feedback (Figure 2b-d). Then I made new designs, i.e. a robust genetic memory element (Figure 8); genetic gates with two inputs and one output were introduced (Figure 5f, 6f), as well as a tristable toggle switch was described in terms of stochastic π-calculus (Figure 3d-f). The last example confirmed the possibility genetic networks with high connectivity – the most important feature of eukaryotes.

Different networks were compared, e.g. the repressilator comprising three ***neg*** genetic elements demonstrated more regular oscillations than the oscillation motif including two proteins (Figure 4c, d). Also, C1 coherent and I1 incoherent feedforward loops were investigated at various conditions. They were so stable that, for instance, C1-FFL behaved like OR logic element in a noise environment at the nominal parameters: $\mathbf{r} = 1.0$, $\boldsymbol{\delta} = 0.001$, $\boldsymbol{\varepsilon}_n = 0.1$, $\boldsymbol{\eta}_n = 0.01$, $\boldsymbol{\varepsilon}_p = 0.01$, $\boldsymbol{\eta}_p = 0.1$ (Figure 5f). However, C1-FFL and I1-FFL produced spontaneous pulses with high amplitude at a very small $\boldsymbol{\varepsilon}_p = 0.00001$ (Figure 5d and Figure 6d).

The coherent and incoherent compositional FFL and FBL circuits generated coherent and decoherent pulses accordingly (Figure 7). These circuits were analyzed systematically at various parameters, their diverse behaviors classified, and attention was given to state III which formed trains of pulses (S1: 15, S2: 15). Variations in parameters were more significant for ***pos*** gates than for ***neg*** ones. In particular, parameter $\boldsymbol{\eta}_p$ playing a role of activator was critical (S2: 13-16). Sometimes distinctions in a network topology and a range of parameters changed the system in a spectacularly unpredictable way. As such, the system demonstrated unusual behavior in state III during a long-time simulation that was impossible to see during a short-time run (S1: 15, S2: 15).

As mentioned, a memory element from four ***neg*** gates was constructed. The circuit follows two very stable states; nevertheless, an intensive external impulse can turn the system from one state to an alternative state. Operations can be repeated many times in both directions (Figure 8d). This model could explain epigenetic effects in living cells. Finally, bifans of various functionality formed particular outputs under controllable inputs. They showed a time-amplitude modulation with diverse patterns of expression (Figure 9). The last design gives me hope that scalable systems with multiple inputs will be able to signal pattern recognition.

The stochastic π-calculus is inherently different from differential equations and allows us to investigate new phenomena. I was able to construct genetic switches and memory elements using stochastic dynamics and network topology variations and avoiding cooperative mechanisms (Figure 3 and Figure 8). My attempts to find memory effects in regulated positive and negative feedback loops were unsuccessful (Supplement 3). This is a discrepancy with the conclusion of Uri Alon (Alon, 2007) who pointed out a memory capability for these kinds of circuits. The disagreement prompted me to think of a more appropriative language for biological systems. It would be very interesting to know about the behavior of these circuits in real conditions *in vivo* and which approach is better.

Conclusion

My job in a 'silicon laboratory' represents the bottom-up scenario of compositional design in terms of SPiM calculus. Most models chosen arbitrarily worked well at standard val-

ues of parameters. The topology and complexity of networks played a significant role in their behavior. New networks emerged easily, sometimes because of duplications and transpositions of the previous ones. These operations resemble natural genetic mechanisms such as vertical and horizontal gene transfer – essential operators of biological evolution (Kuznetsov, 2007). Some variations in parameters are similar to 'point mutations' leading to an optimization (adaptation) of the system to a desirable pattern of behavior. Future investigations will shed light on which formalism – the stochastic π-calculus or Ordinary Differential Equations – is better able to describe biological entities.

Aknolegements

The author is grateful to Andrew Phillips, Vladik Avetisov, Genaro Juarez Martinez, and an anonymous referee for suggestions and to Bert Schnell for his help with the manuscript.

References

1. Alon U (2007) An Introduction to Systems Biology: Design Principles of Biological Circuits. Chapman & Hall/Crc Mathematical and Computational Biology. » CrossRef » PubMed » Google Scholar
2. Blossey R, Cardelli L, Phillips A (2006) A Compositional Approach to the Stochastic Dynamics of Gene Networks, Transactions. Computer Science 3939: 99-122. » CrossRef » PubMed » Google Scholar
3. Blossey R, Cardelli L, Phillips A (2008) Compositionality, Stochasticity and Cooperativity in Dynamic Models of Gene Regulation. HFSP J 2: 17-28. » CrossRef » PubMed » Google Scholar
4. Cardelli L (2009) Artificial Biochemistry. Computer Science 429-462. » CrossRef » PubMed » Google Scholar
5. Gillespie D (1977) Exact stochastic simulation of coupled chemical reactions. J Phys Chem 81: 2340-2361. » CrossRef » PubMed » Google Scholar
6. Kashtan N, Alon U (2005) Spontaneous evolution of modularity and network motifs. Proc Natl Acad Sci 102: 13773-13778. » CrossRef » PubMed » Google Scholar
7. Kashtan N, Noor E, Alon U (2007) Varying environments can speed up evolution. Proc Natl Acad Sci 104: 13711-13716. » CrossRef » PubMed » Google Scholar
8. Kreimer A, Borenstein E, Gophna U, Ruppin E (2008) The evolution of modularity in bacterial metabolic networks. Proc Natl Acad Sci 105: 6976-6981. » CrossRef » PubMed » Google Scholar
9. Kuznetsov A (2007) Evolution by Communication: A Revision of Sperm-Mediated Gene Transfer. Frontiers in the Convergence of Bioscience and Information Technologies 322-326. » CrossRef » PubMed » Google Scholar
10. Kuznetsov A (2009a) Modularity and distribution of sulfur metabolism genes in bacterial populations: search and design. J Comput Sci Syst Biol 1573-1634. » CrossRef » PubMed » Google Scholar
11. Kuznetsov A (2009b) Synthetic Biology as a proof of Systems Biology. Handbook of Research on Systems Biology Applications in Medicine Ed Andriani Daskalaki IGI Global 97-115. » CrossRef » PubMed » Google Scholar
12. Milner R (1992) Functions as Processes. Computer Science 2: 119-141. » CrossRef » PubMed » Google Scholar
13. Milner R (1999) Communicating and Mobile Systems: the π-Calculus. Cambridge University Press UK. » CrossRef » PubMed » Google Scholar
14. Parter M, Kashtan N, Alon U (2007) Environmental variability and modularity of bacterial metabolic networks. BMC Evol Biol 7: 169-177. » CrossRef » PubMed » Google Scholar
15. Phillips A (2007) The SPiM Language (Version 0.05). Available from http://research.microsoft.com/en-us/um/people/aphillip/spim/Language.pdf
16. Phillips A (2009a) A Visual Process Calculus for Biology, Symbolic Systems Biology: Theory and Methods. Jones and Bartlett Publishers In Press Available from http://research.microsoft.com/en-us/people/aphillip/
17. Phillips A (2009b) An Abstract Machine for the Stochastic Bioambient calculus. Electronic Notes in Theoretical Computer Science 227: 143-159. » CrossRef » PubMed » Google Scholar
18. Phillips A, Cardelli L (2007) Efficient, correct simulation of biological processes in the stochastic Pi-calculus. Comp Methods Syst Biol 4695: 184-199. » CrossRef » PubMed » Google Scholar
19. Poli R, Langdon WB, Mcphee NF (2008) A field guide to genetic programming. (With contribution by Koza JR) GPBib. » CrossRef » PubMed » Google Scholar
20. Priami C, Regev A, Silverman W, Shapiro E (2001) Application of stochastic process algebras to bioinformatics of molecular processes. Information Processing Letters 80: 25-31. » CrossRef » PubMed » Google Scholar
21. SPiM Player (Version 1.13). Available on 20.10.2008 from http://research.microsoft.com/research/downloads/details/992f59a0-c8c2-40bc-ab25-34516cf132c9/details.aspx
22. Thompson A (1998) Hardware Evolution: Automatic design of electronic circuits in reconfigurable hardware by artificial evolution. Springer-Verlag. » CrossRef » PubMed » Google Scholar
23. Watson RA (2006) Compositional Evolution: The Impact of Sex, Symbiosis, and Modularity on the Gradualist Framework of Evolution. Vienna Series in Theoretical Biology: A Bradford Book. » CrossRef » PubMed » Google Scholar

see Appendix and Supplement 4