A compositional approach to the stochastic dynamics of gene networks

Ralf Blossey*, Luca Cardelli†, and Andrew Phillips†

*Interdisciplinary Research Institute, F-59652 Villeneuve d'Ascq, France; †Microsoft Research, CB3 0FB Cambridge, United Kingdom

We propose a compositional approach to the dynamics of gene regulatory networks based on the stochastic π-calculus, and develop a representation of gene network elements which can be used to build complex circuits in a transparent and efficient way. To demonstrate the power of the approach we apply it to several artificial networks, such as the repressilator and combinatorial gene circuits first studied in Combinatorial Synthesis of Genetic Networks (1). For two examples of the latter systems, we point out how the topology of the circuits and the interplay of the stochastic gate interactions influence the circuit behavior. Our approach may be useful for the testing of biological mechanisms proposed to explain the experimentally observed circuit dynamics.

Within the last years a general consensus has emerged that noise and stochasticity are essential building elements of gene regulatory networks. A quantitative understanding of their role is thus needed to understand gene regulation. Regulatory functions can indeed work to eliminate stochastic effects (2), or to even exploit them (3).

In line with new experimental techniques to measure and quantify such behavior, efficient ways to model and simulate gene networks need to be developed, which are currently lacking. Simulations based on differential equations for the concentrations of the various biomolecules, the long-time standard of modeling in biochemical systems are not well suited for this purpose, except in particular cases. Stochastic effects, which are typically important when molecule numbers are small, are difficult to build into such approaches, and the resulting stochastic equations are time-consuming to simulate. In addition, differential equation models are inherently difficult to change, extend and upgrade, as changes of network topology may require substantial changes in most of the basic equations.

In this paper, we follow a different route. It has recently emerged within computer science in the context of process calculi, and their applications to biological systems. Process calculi (4) are essentially programming languages designed to describe concurrent, interactive systems such as mobile communication networks. Among the various process calculi, π-calculus is one of the best studied because of its compactness, generality, and flexibility. Stochastic variants have appeared recently that address biochemical modeling (5); they have been used to model molecular interactions (7) and compartments (8,9). A remaining challenge is to model gene networks, to fully demonstrate the flexibility of process calculi, and to eventually support the integration of molecular, gene, and membrane networks in a single framework.

Here, we introduce process calculi by example, in the context of gene networks; technical details of the approach can be found in the Supplementary Online Material (10) and the references given there. Modeling with process calculi is very much like programming, and yet it is mathematically precise. It is carried out in concurrent, stochastic programming languages that can easily support very complex and detailed models in a modular ("compositional") way, where separate program units correspond to separate biochemical components.

Our purpose here is in part tutorial: we aim to show that we can do things *simply* to start with, and already get interesting insights. More sophisticated models can be built when needed; e.g. see (11) for a detailed molecular model of transcription-translation in phage lambda done in this style. The paper is structured as follows: in the Methods sections we explain how to represent gene network elements as processes in the stochastic π-calculus and how to execute them. We then apply this representation to model gene networks of increasing complexity, and study some of their behavior. In particular, we address the repressilator circuit (12) and two of the (still controversial) examples of combinatorial circuits first discussed in (1).

Methods I: Modeling Gene Network Elements

Nullary Gates. We begin by modeling genes that have constitutive transcription but no regulatory control. We focus on the *actions* that are involved in the functioning of genes and molecular components. The generic term *process* is used for any mechanism performing actions and thus progressing through distinct states.

A nullary-input gate (Fig. 1), given by a process written *null*(*b*), has a single parameter *b* that represents its transcription product; it takes no input from the environment. The initial action performed by such a gate is a stochastic delay, τ_ε, where τ is a symbol indicating delay and ε is a given rate constant (we always use exponential distributions characterized by rate constants). After such a delay action, the original process *null*(*b*) becomes (i.e., changes state to) two separate processes in parallel ("|"): *tr*(*b*) and *null*(*b*). The second process is a copy of the original process *null*(*b*), which was consumed when performing its initial action. The first process, *tr*(*b*), described shortly, represents a molecule of a transcription factor for a binding site *b* on some gene. All together, the *null*(*b*) process is defined as $\tau_\varepsilon.\ (tr(b) \mid null(b))$. A stochastic simulation of a *null*(*b*) process on its own produces multiple copies of *tr*(*b*) at stochastic time intervals ε, with exactly one copy of *null*(*b*) being preserved.

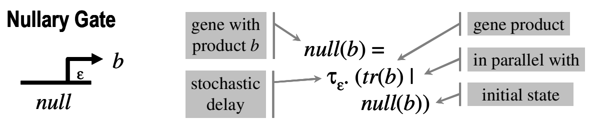


Fig. 1. A gene, *null*(*b*) with constitutive transcription, but no regulation (nullary). The product is a translated protein, *tr*(*b*) that attaches to a binding site *b* on some other gene; the definition of *tr*(*b*) is given later. The definition of *null*(*b*) says that this gate waits for a stochastic *delay* ('τ') of rate ε, and *then* ('.') evolves into two processes in *parallel* ('|'); one is *tr*(*b*), and the other again *null*(*b*), the initial state.

Gene Products. We now describe the transcription factor *tr*(*b*) (Fig. 2), introducing the process calculus notions of interaction and nondeterministic choice. Except for delays τ, which happen autonomously, any action that a process performs must happen in conjunction with a complementary action performed by another process. The simultaneous occurrence of complementary actions is an *interaction*, e.g. between two molecules, or between a transcription factor and a promoter site. An action can be *offered* at any time, but only complementarily offered actions can result in actual interactions. For an interaction site, or *channel*, *b*, such complementary actions are conventionally called *input on b* (written '?*b*'), and *output on b* (written '!*b*'). An interaction is in general a *communication* where *messages* are exchanged from output actions to input actions. But in this paper we need only consider *synchronization* interactions that only exchange a *signal* (which can be considered as a trivial message). Hence, ?*b* and !*b* are complementary actions that can exchange a signal between them and allow two corresponding processes to change state.

The transcription factor *tr*(*b*) offers a choice of two actions; one is an output action !*b*, representing interaction with a binding site, and the other is a delay τ, followed by degradation. These two actions are in a stochastic race, indicated by '+': *b* has (implicitly defined with it) a fixed associated rate *r*, and τ has a specific rate δ. If !*b* wins the race, it means that an interaction has occurred with an input action ?*b* offered elsewhere, and the process returns to the initial state, *tr*(*b*). If τ wins the race, however, the following state is 0: the inert process that never performs any actions.

All together, *tr*(*b*) is defined as (!*b*. *tr*(*b*)) + (τ_δ.0), which means that *tr*(*b*) has the potential to interact multiple times with promoter sites, but each time (and particularly if no promoter site is available) it has a chance to degrade. Without interactions with binding sites, a fixed population of transcription factors will simply exponentially degrade. If the population is being replenished, then a stable level may be found between production and degradation.

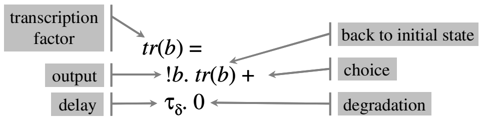


Fig. 2. A transcription factor *tr*(*b*) makes a *stochastic choice* ('+') between either binding to an available promoter site *b* by an *output action* ('!*b*'), or *delaying* ('τ') with rate δ. In the first case, the output action interacts with a corresponding input action at a promoter site *b*, and *then* ('.') the transcription factor returns to its initial state *tr*(*b*), ready to interact again. In the second case, the transcription factor degrades to the inert process ('0').

Unary Gates. We now consider gates with simple regulation. A *neg*(*a*,*b*) gate has a promoter site *a* with negative regulation (inhibition), and a product *b*.

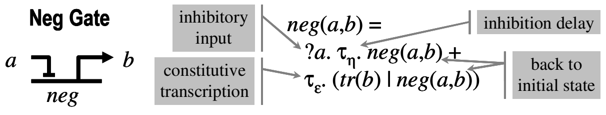

Neg Gate
a
b
neg
inhibitory input
constitutive transcription
$neg(a,b) =$
$?a.\ \tau_\eta.\ neg(a,b)\ +$
$\tau_\varepsilon.\ (tr(b) \mid neg(a,b))$
inhibition delay
back to initial state

Fig. 3. A gene gate with inhibitory control, *neg*(*a*,*b*) makes a stochastic choice ('+') between constitutive transcription and inhibitory stimulation. The constitutive transcription case (bottom line) is exactly as in Fig. 1, but this time it is in a race with a stimulus. If an interaction happens with the input action '?*a*', then the gate enters a stochastic delay ('τ_η'), during which the gate is inhibited, and then returns to the initial state.

The *neg*(*a*,*b*) gate (Fig. 3) has a subprocess that is essentially identical to the *null*(*b*) gate, i.e., it provides constitutive transcription. However this subprocess is now in a stochastic race with a subprocess ?*a*. τ_η. *neg*(*a*,*b*). That is, it is in a race with a promoter binding, ?*a*. If the promoter component wins the race (by interacting with a transcription factor *tr*(*a*)), the + choice is taken on the promoter side, and the whole process becomes τ_η. *neg*(*a*,*b*). In this state, the gate is stuck performing a stochastic delay τ_η, i.e., it is inhibited, after which it goes back to be *neg*(*a*,*b*).

The *pos*(*a*,*b*) gate (Fig. 4) has a promoter site *a* with positive regulation (stimulation), and a product *b*. It is similar to the *neg* gate, but instead of an inhibition delay, we have a transcription delay followed by stimulated production of *tr*(*b*).

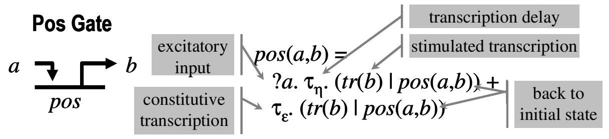

Pos Gate
a
b
pos
excitatory input
constitutive transcription
$pos(a,b) =$
$?a.\ \tau_\eta.\ (tr(b) \mid pos(a,b))\ +$
$\tau_\varepsilon.\ (tr(b) \mid pos(a,b))$
transcription delay
stimulated transcription
back to initial state

Fig. 4. A gene gate with excitatory control, *pos*(*a*,*b*). This is almost identical to *neg*(*a*,*b*), but the input stimulus '?*a*' is followed by the production of *tr*(*b*) instead of an inhibitory delay.

Methods II: The Stochastic π-Calculus Execution Model

Simulation Language. We have seen how a biological system can be modeled in the stochastic π-calculus, by representing each component of the system as a process *P* that precisely describes what the component can do. To summarize, the most basic process form is a choice $\Sigma = P_1 + \ldots + P_n$ between zero or more outputs !*x*(*n*), inputs ?*x*(*m*), and delays τ that the component can perform (*n* is the output message and *m* is the input variable). Two components *P* and *Q* can be combined together using parallel composition *P*|*Q*. Channels can be established to allow the components to interact by complementary inputs and outputs. Once a biological system has been modeled using these basic components, the model can be stochastically simulated in order to predict the evolution of the system over time. In this paper, the simulations were obtained using the Stochastic Pi Machine (SPiM), which is described in (13).

Simulator. The Stochastic Pi Machine simulates a given process *P* by first converting the process to a corresponding simulator data structure, consisting of a list of components $A = \Sigma_1, \ldots, \Sigma_M$. The resulting list is then processed by the simulator, by first using a function *Gillespie*(*A*) to stochastically determine the next interaction channel *x* and the corresponding reaction time τ. Once an interaction channel *x* has been chosen, the simulator uses a *selection operator* to randomly select from the list *A* a component of the form Σ+?*x*(*m*).*P* containing an input on channel *x*, and different component of the form Σ' +!*x*(*n*).*Q* containing an output on *x*. The selected components can then interact by synchronizing on channel *x*, and the processes *P* (with the input variable *m* replaced by *n*) and *Q* are added to the remainder of the list. The simulator continues processing the list in this way until no more interactions are possible.

The function *Gillespie*(*A*) is based on (14), which uses a notion of *channel activity* to stochastically choose a reaction channel from a set of possible channels. The activity of a reaction channel corresponds to the number of possible combinations of reactants on the channel, and channels with a high activity and a fast reaction rate have a higher probability of being selected. A similar notion of activity is defined for the Stochastic Pi Machine, where $Act_x(A)$ denotes the number of possible combinations of inputs and outputs on interaction channel *x* in a list of components *A*:

$$Act_x(A)=(In_x(A)*Out_x(A))-Mix_x(A)$$

$In_x(A)$ and $Out_x(A)$ are defined as the number of available inputs and outputs on interaction channel *x* in *A*, respectively, and $Mix_x(A)$ is the sum of $In_x(\Sigma_i)\times Out_x(\Sigma_i)$ for each component Σ_i in *A*. The formula takes into account the fact that an input and an output in the same component cannot interact, by subtracting $Mix_x(A)$ from the product of the number of inputs and outputs on *x*.

The Stochastic Pi Machine has been formally specified in (13), and the specification has been proved to correctly simulate all possible π-calculus processes. The simulator has also been used to simulate a wide variety of chemical and biological systems. In particular, many of the benchmark examples that were used to validate the Gillespie algorithm (14) have been modeled as π-calculus processes and correctly simulated in SPiM. Details of the simulation results are available from (10).

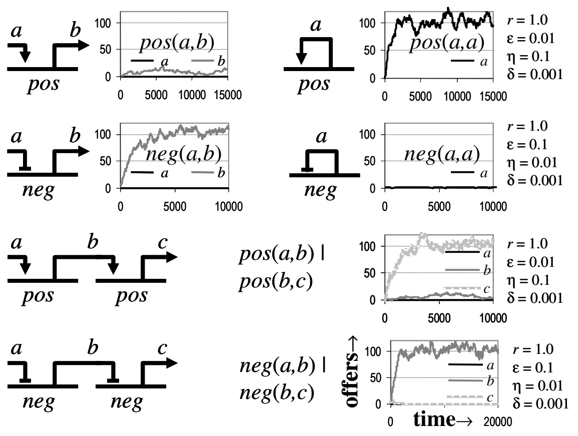


Fig. 5. Compositions of gates represent circuits (left) that exhibit behaviors (right). The channels *a,b,c* are declared separately (not shown) along with their associated stochastic interaction rates. In all simulations, the common rate *r* for *a,b,c* is set to 1.0. The other chosen rates are as indicated in the individual simulations. The vertical axis is the number of outstanding *offers* of communication: for a channel *a* we may plot output offers !*a* or input offers ?*a*. In all cases above, the networks get started by constitutive transcription only. All the plots are of individual simulator runs.

Interaction-Oriented Simulation vs. Reaction-Oriented Simulation. The Gillespie algorithm was originally used to simulate a set of reaction equations expressed in terms of chemical reactants and products, and the results of a simulation were plotted as the quantity of each chemical species versus time. In contrast, the π-calculus does not describe an equation for each type of reaction, but instead describes the behavior of each component in terms of the inputs and outputs it can perform on a set of interaction channels. This gives rise to an interaction-oriented model, as opposed to a reaction-oriented model, in which a reactant is defined as an input or output on a given interaction channel. Once the notion of a reactant has been defined in this way, the Gillespie algorithm can be directly applied to a given π-calculus model of the biological system. The corresponding simulation results can be plotted as the quantity of each reactant versus time.

Results

Simple Circuits. We now have all the network components we need, at least for gates with one input. Gates with *n* inputs can be defined similarly, to form a larger library of components. Building gene circuits now consists of providing interaction channels, with associated interaction rates, connecting the various gates. If we write, e.g., *pos*(*a*,*b*) | *neg*(*b*,*c*), the *pos* process will offer an output action !*b*, through *tr*(*b*), and the *neg* process will offer an input action ?*b*. Hence the shared channel *b*, given to both *pos* and *neg* as a parameter, can result in repeated interactions between the two processes, and hence in network connectivity.

The simplest circuits we can build are single gates interacting with themselves in a feedback loop, like *pos*(*a*,*a*). In absence of any stimulus on *a*, *pos*(*a*,*a*), must choose the constitutive transcription route and evolve into *tr*(*a*) | *pos*(*a*,*a*), where now *tr*(*a*) can stimulate *pos*(*a*,*a*) at a faster rate, and possibly multiple times. Depending on the production and degradation rates, a stable high level of *tr*(*a*) may be reached. Similarly *neg*(*a*,*a*) can stabilize at a low quantity of *tr*(*a*) where degradation of *tr*(*a*) balances inhibition. A convenient high-signal level of about 100 is maintained in our examples by appropriate parameters (Fig. 5).

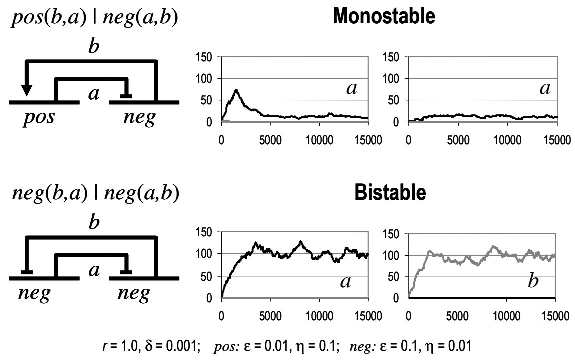


Fig. 6. Monostable and bistable feedback loops.

The combination *pos*(*b*,*a*) | *neg*(*a*,*b*) (Fig. 6) is a self-inhibition circuit, like *neg*(*a*,*a*), and it similarly has a stable output. But now there are two separate products, *tr*(*a*) and *tr*(*b*), so the system (again in absence of any stimulus) can stochastically start with a prevalence of *tr*(*a*) or a prevalence of *tr*(*b*): this can be seen at the beginning of the two plots, before stabilization.

The combination *neg*(*b*,*a*) | *neg*(*a*,*b*) (Fig. 6) is a bistable circuit, which can start up in one state or another, and (usually) stay there.

Repressilator. The well-known repressilator circuit (12), consisting of three *neg* gates in a loop, is an oscillator. We compare here three different degradation models, aiming to justify somewhat our initial definition for *tr*(-). In the first model (Fig. 7 A), each transcription factor interacts exactly once, and only then it disappears. The repressilator circuit oscillates nicely, but without stochastic degradation, the plots appear very "mechanical"; the quantities of products grow at each cycle, because products do not disappear unless they interact. In the second model (Fig. 7 B), each transcription factor interacts exactly once, or can degrade. Again the plots look mechanical, but the stochastic degradation defines a stable level of product. The third model (Fig. 7 C), with multiple interactions and stochastic degradation, is more realistic and gives more convincing plots. See (10) for the simulator script.

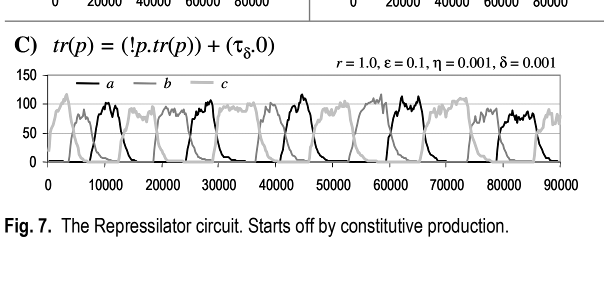


Fig. 7. The Repressilator circuit. Starts off by constitutive production.

The progressive refinement of the definition of *tr*(-), provides an illustration of how one can easily play with process descriptions to find models that show a balance between simplicity and realism. A further step could be to model both attachment and detachment of transcription factors, and then to model both transcription and translation.

Network properties: Oscillation. It is instructive to take a "systems" approach and see what the gate parameters described earlier mean in the context of networks of gates. In the case of the repressilator we can see that the constitutive rate (together with the degradation rate) determines oscillation amplitude, while the inhibition rate determines oscillation frequency. Fig. 8 shows the variation of ε and η from their values in Fig. 7 C); note the differences in scale.

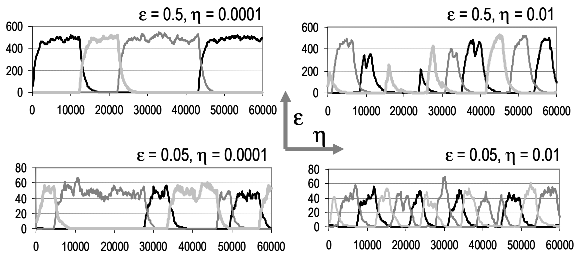


Fig. 8. Repressilator frequency and amplitude, regulated by η and ε. *Cf.* Fig. 7(C).

Moreover, we can view the interaction rate *r* as a measure of the volume (or temperature) of the solution; that is, of how often transcription factors bump into gates. Fig. 9 shows that the oscillation frequency and amplitude remain unaffected in a large range of variation of *r* from its value in Fig. 7 C). Note that *r* is in stochastic race against δ in *tr*, and δ is always much slower.

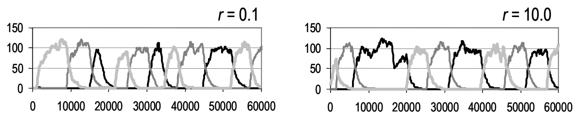


Fig. 9. Repressilator stability to changes in *r* (volume/temperature). *Cf.* Fig. 7(C).

Network properties: Fixpoint. We now discuss a network property that becomes important in later analysis. Fig. 10 plots signals flowing through a sequence of *neg* gates with parameters as in Fig. 7 C, except for η, the inhibition delay. On the left, the signals are alternating between high (*b*,*d*) and low (*a*,*c*,*e*). As η is increased, shown from left to right, the gates behave less and less like boolean operators, but the signals remain separate.

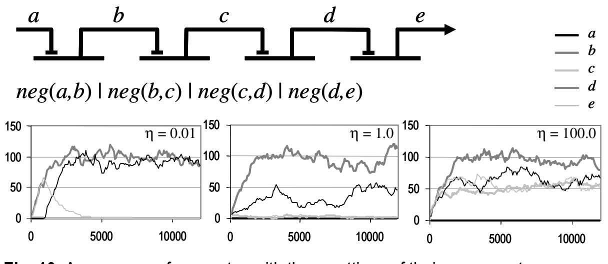


Fig. 10. A sequence of *neg* gates with three settings of their η parameter.

Fig. 11 shows the same circuit, except for a self feedback on the head gate. With low inhibition delay η (i.e. ineffective feedback) the system is unstable (left). But soon after, as we increase η, the self feedback flattens *all* signals downstream to a common low level (middle). The signals remain at a common level over a wide range of η, although this level is raised by increasing η (right).

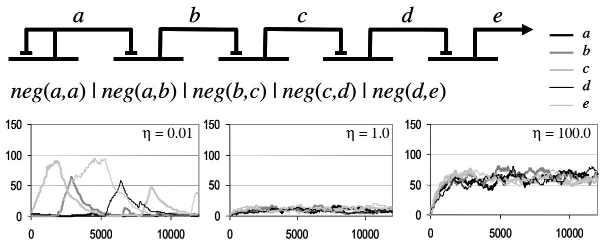
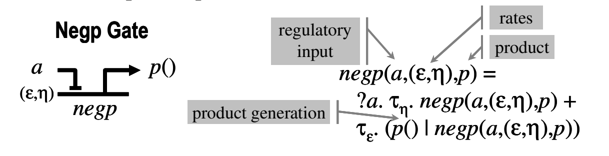


Fig. 11. The effect of head feedback on a sequence of *neg* gates.

This behavior is self-regulating, and can be explained as follows. The head feedback naturally finds a fixpoint where gate input equals gate output (unless it oscillates). If the next gate has the same parameters, its output will then also equal its input, and so on down the line: all the gates will be at the same fixpoint. Different values of η and different gate response profiles may change the fixpoint level, but not its fundamental stability.

Combinatorial Circuits. As examples of non-trivial combinatorial networks and their stochastic simulation, we finally examine the artificial gene circuits described by Guet *et al.* (1). Most of those circuits are simple combinations of inhibitory gates, however, it was found that in some of the circuits subtle (and partially still not understood) behavior arises from their connectivity; we focus particularly on two of these cases. The circuits are exercised by varying two inputs (two chemicals in the environment: *aTc* and *IPTG*) that act as repressors for some of the internal signals. A single output is observed in the experiments: the concentration of a fluorescent protein product (*GFP*).



Negp Gate
a
(ε,η)
p()
negp
regulatory input
rates
product
product generation
$negp(a,(\varepsilon,\eta),p) =$
$?a.\ \tau_\eta.\ negp(a,(\varepsilon,\eta),p)\ +$
$\tau_\varepsilon.\ (p()\ |\ negp(a,(\varepsilon,\eta),p))$

Fig. 12. A *neg* gate with parametric product *p*.

In order to build up the different combinatorial networks easily, we begin with a version of the *neg* gate that is more flexibly parameterizable. We call it *negp*, and it has the property that, if *s* represents the rates used in the *neg* gate, then *negp*(*a*,*s*,*tr*(*b*)) = *neg*(*a*,*b*), hence *neg* is a special case of *negp*. The rates for inhibition and constitutive translation are passed as a pair *s*=(ε,η), in the second parameter. The third parameter fully encapsulates the gate product, so the gate logic is independent of it[1].

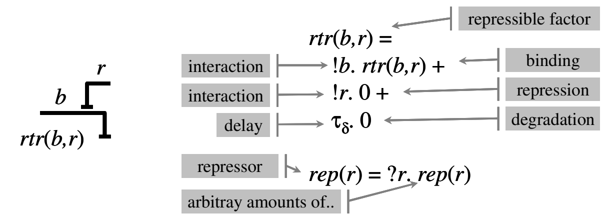
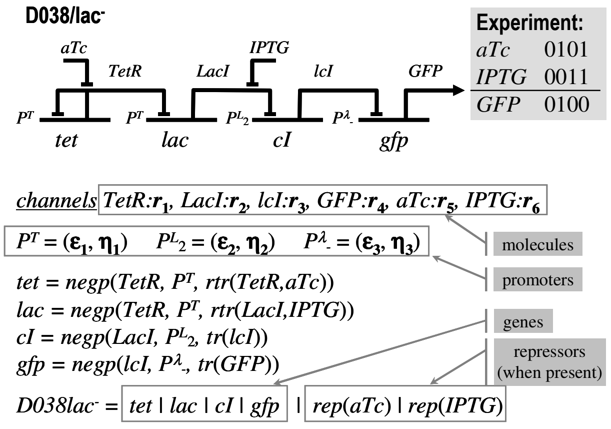


Fig. 13. Repressible transcription factors.

In addition to the old transcription factors *tr*(*b*), binding to a site *b*, we now need also transcription factors that can be repressed: *rtr*(*b*,*r*). These have three possible behaviors: binding to a site *b*, being neutralized via a site *r*, and degrading. The repression is

[1] More technically, if we set *pb*() = *tr*(*b*) (*pb* is the process that when invoked with no arguments, invokes *tr* with argument *b*), then we have *negp*(*a*,*s*,*pb*) = *neg*(*a*,*b*); we write *negp*(*a*,*s*,*tr*(*b*)) as an abbreviation, skipping the intermediate definition of *pb*.

performed by a process *rep*(*r*) that, if present, "inexhaustibly" offers ?*r*.



Experiment:	
aTc	0101
IPTG	0011
GFP	0100

channels $TetR{:}r_1$, $LacI{:}r_2$, $lcI{:}r_3$, $GFP{:}r_4$, $aTc{:}r_5$, $IPTG{:}r_6$

$P^T = (\varepsilon_1, \eta_1)$ $P^{L}_2 = (\varepsilon_2, \eta_2)$ $P^{\lambda}_- = (\varepsilon_3, \eta_3)$

tet = *negp*(*TetR*, P^T, *rtr*(*TetR*,*aTc*))
lac = *negp*(*TetR*, P^T, *rtr*(*LacI*,*IPTG*))
cI = *negp*(*LacI*, P^{L}_2, *tr*(*lcI*))
gfp = *negp*(*lcI*, P^{λ}_-, *tr*(*GFP*))
D038lac$^-$ = *tet* | *lac* | *cI* | *gfp* | *rep*(*aTc*) | *rep*(*IPTG*)

Fig. 14. D038.

We can now describe the circuits from (1) by simple combinations of *negp*, *tr*, *rtr*, and *rep* components. All the other names appearing here, such as *TetR, aTc*, etc., which glue the network together, are just pure *interaction names*: they have no structure or behavior of their own, except for being used in complementary input and output actions.

Intuitive Boolean analysis of one of the still controversial circuits, D038, in Fig. 14 would suggest either oscillation (*GFP*=0.5 on average), or *GFP*=1, contrary to experiment[2]. The fixpoint effect, however, suggests an explanation for the output in absence of repressors, whereby all signals, including *GFP* are driven to a low fixpoint. Adding *aTc* breaks that equilibrium, driving *TetR* fully to 0, and hence *GFP* to 1. In all cases, adding *IPTG* drives *LacI* to 0 and hence *GFP* to 0. Fig. 15 shows the simulation results for the program in Fig. 14.

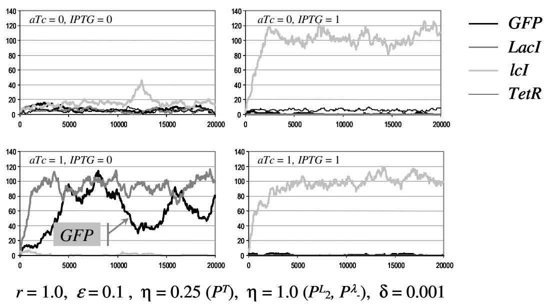
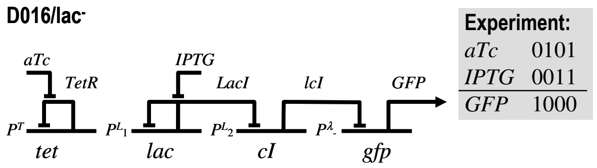


Fig. 15. D038 simulations.

In a very similar fashion we can code another peculiar circuit, D016, shown in Fig. 16. This circuit is perplexing because addition of *aTc*, affecting an apparently disconnected part of the circuit, changes the *GFP* output. In (17) it is suggested that this may be caused by an overloading of the degradation machinery, due to an overproduction of *TetR* when *aTc* is present, which might decrease the degradation rate of the other proteins. But even in absence of *aTc* and *IPTG*, it is surprising that *GFP* is high (about 50% of max (15)): this seems to contradict both simple boolean analysis and our fixpoint explanation.



Experiment:	
aTc	0101
IPTG	0011
GFP	1000

Fig. 16. D016

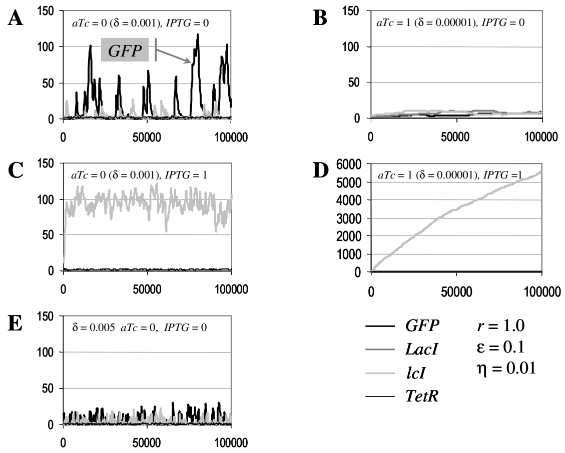


Fig. 17. D016 simulations

A possibility to rationalize this behavior is to assume that P^{L}_1-*lac* is operating in an instability region. A closer examination of the instability region of our fixpoint circuit (Fig. 10 bottom left) shows that, while the first signals in the sequence (*a*,*b*) are kept low, the subsequent signals (*c*, corresponding to *GFP* in D016, and *d,e*) all spike frequently. This may give the appearance, on the average, of high levels of *GFP*, matching the first column of the D016 experiment. Moreover, the region of instability is very sensitive to degradation levels: *GFP* levels can be brought down both by increasing degradation by a factor of 5 (because this brings the circuit back into the fixpoint regime) or by decreasing degradation by a factor of 1000 (so that there are enough transcription factors to inhibit all gates). In Fig. 17 we begin by placing D016 in the instability region, with *GFP* spiking (A). Then, adding *aTc* while reducing degradation suppresses all signals (B). Adding *IPTG* results in no *GFP* (C,D); moreover, reduced degradation causes overproduction (D). Even increased degradation (E) can result in no *GFP*. A proper biological explanation of the behavior of D016 has not been obtained yet, but analysis such as the above indicates that circuits with head feedbacks are extremely sensitive to a number of conditions, and that many surprising behaviors are possible (18).

Conclusions

In this paper we have demonstrated how stochastic simulations of gene circuits can be built in a compositional way by employing the stochastic π-calculus. Compositionality is illustrated, for example, by our treatment of the repressilator circuit: the definition of the *neg* gate could be left unchanged when the definition of the transcription factor *tr* was refined. Our approach is mechanistic in the sense that we model discrete components and

[2] In absence of repressors, the experimentally observed *GFP* is 0 (meaning no detectable signal), hence, by tracing boolean gates backwards, *lcI*=1, and *LacI*=0, and *TetR*=1. But by self-loop *TetR*=1 implies *TerR*=0, so the whole circuit, including *GFP* should be oscillating and averaging *GFP*=0.5. As an alternative analysis, consider the level of *TetR* (which is difficult to predict because it is the result of a negative self-feedback loop). Whatever that level is, and whether or not *aTc* is present, it must equally influence the *tet* and *lac* genes, since the promoters are the same (P^T). The option, *TetR*=*LacI*=1 gives *GFP*=1. Suppose instead *TetR*=*LacI*=0, then *lcI*=1, and *GFP*=0 as observed. But in that situation, with *TetR*=0, *aTc* should have no influence, since it can only reduce the level of *TetR*. Instead, *aTc* somehow pushes *GFP* to 1.

derive the effects of their interactions. This differs from modeling changes in concentration levels whose mechanistic causes are undetermined (as was done in (17)). On the other hand, the mechanistic aspect is balanced by the fact that our approach is abstract, as it allows a considerable flexibility in the level of detail with which components and their interactions are described (see Supplementary Online Material for further illustration). While the level of detail adopted here may be considered coarse and qualitative, it can be easily and compositionally refined to match available knowledge.

Apart from these analytical and conceptual advantages in building up the different circuits, we stress that the ease of use of the compositional approach in combination with stochastic simulations is particularly useful for hypothesis testing. It can build on available knowledge, but the outcome of the stochastic simulations of the interacting components yields a highly non-trivial check of expectations. By comparison, Boolean analysis or intuitive ideas are obviously too naïve and thus can easily be misleading.

This latter point is illustrated by our findings for the controversial combinatorial circuits we discussed last in the paper. In particular, we observe a significant dependence of the system behavior on promoter strengths and degradation processes. Promoter strengths are generally poorly known, and even qualitative relationships between the different promoters are missing (18).

To conclude, we believe that the compositional approach we propose for the formulation of stochastic models of gene networks will allow a useful path for more detailed, quantitative studies of regulatory mechanisms, and in particular for the testing of hypotheses of complex system behavior. It may be considered as one step towards the development of flexible languages and simulation tools for computational biology, for which a need has recently been expressed by several biologists (19-21).

1. Guet, C.C., Elowitz, M.B., Hsing, W. & Leibler, S. (2002) *Science* **296** 1466-1470.
2. Thattai, M. & van Oudenaarden, A. (2001) *Proc. Nat. Acad. Sci.* **98**, 8614- 8619.
3. Paulsson, J., Berg, O.G. & Ehrenberg M. (2000) Proc. Nat. Acad. Sci. **97**, 7148-7153.
4. Milner, R. (1999) Communicating and Mobile Systems: The Pi-Calculus. Cambridge University Press.
5. Priami, C., Regev, A., Shapiro, E. & Silverman, W. (2001) *Information Processing Letters* **80** 25-31.
6. Regev, A. (2002) Ph.D. Thesis, Tel Aviv University.
7. Regev, A. & Shapiro, E. (2002) *Nature* **419** 343.
8. Regev, A., Panina, E.M., Silverman, W., Cardelli, L. & Shapiro, E. (2004) *Theoretical Computer Science*, **325(1)** 141-167.
9. Cardelli, L. (2004) *Computational Methods in Systems Biology*. Springer. 257-278.
10. Supplementary Online Material to this paper.
11. Kuttler, C.& Niehren, J. (2005), submitted.
12. Elowitz, M.B., Leibler. S. (2000) *Nature* **403** 335-338.
13. Philips, A. & Cardelli, L., (2005), submitted.
14. Gillespie, D. (1977) *J. Chem. Phys*. **81** 2340-2361.
15. Guet, C.C., *personal communication*.
16. Wigler, M. & Mishra, B. (2002) *Science* **296** 1407-1408.
17. Mao, L. & Resat, H. (2004) *Bioinformatics* **20** 2258-2269.
18. Ronen, M., Rosenberg, R., Shraiman, B.I. & Alon, U. (2002) *Proc. Nat. Acad. Sci*. **99** 10555-10560.
19. Brenner, S. (1995) *Curr. Biology* **5** 332.
20. Bray, D. (2001) *Nature* **412** 863.
21. Lazebnik, Y. (2002) *Cancer Cell* **2** 179-182.