Hybrid Semantics for Stochastic π-calculus

Luca Bortolussi[1] and Alberto Policriti[2]

[1] Dept. of Mathematics and Computer Science, University of Trieste, Italy.
luca@dmi.units.it
[2] Dept. of Mathematics and Computer Science, University of Udine, and
Applied Genomics Institute, Udine Italy
policriti@dimi.uniud.it

Abstract. We put forward a method to map stochastic π-calculus processes in chemical ground form into hybrid automata, a class of dynamical systems with both discrete and continuous evolution. The key ingredient is the separation of control and molecular terms, which turns out to be related to the conservation properties of the system.

1 Introduction

Systems biology is a fertile research area in which experimental techniques are coupled with mathematical modeling in order to understand the complex dynamics within cells [16]. In this context, both mathematical and computational tools play an important role: biological systems must be described in a precise mathematical framework, usually ordinary differential equations or stochastic processes, and the obtained models must then be analyzed. However, due to their intrinsic complexity, computational techniques like simulation must be used.
The contributions of Computer Science, on the other hand, are not just restricted to the computational analysis of models, but also related to their description by means of suitable formal languages, offering the features of *compositionality* and *model reusability* [19].
These languages usually belong to the domain of stochastic process algebras (SPA), which have a semantics defined in terms of Continuous Time Markov Chains [14, 21]. Different SPA have been used in biological modeling; here we recall stochastic π-calculus [18], PEPA [6], and stochastic Concurrent Constraint Programming (sCCP, [5]). An important issue in using these languages is that the resulting models are automatically interpreted as stochastic processes, hence they can be simulated and analyzed using common techniques, like the celebrated Gillespie's algorithm [11]. Actually, when just biochemical reactions are involved, the stochastic process defined by these SPA coincide with the canonical one, as given by the chemical master equation [11, 21]. In this approach, the system is described *microscopically*, counting the number of molecules of each species.
Simulation of stochastic processes, however, can be computationally too demanding, especially when large populations of chemical species are present in the system. In this case, a better strategy is that of using models based on ordinary differential equations, approximating the number of molecules as a continuous

quantity. In order to achieve this goal, PEPA, stochastic π-calculus, and sCCP have been equipped with a semantics based on ODE's [3, 15, 8].

This sort of approximation, however, is not appropriate for models having small populations and for models containing inherently discrete entities like genes (genes are usually present in a single copy in a cell, and they are neither produced nor degraded). Actually, there are several well-known examples in which the stochastic model and its continuous approximation show a radically different behavior [11, 2]. In order to deal with some of these cases, in [4] we introduced a hybrid approximation for sCCP, in which some entities of the system are kept discrete, while others are approximated as continuous. Essentially, we defined a mapping from sCCP to hybrid automata, which are mixed discrete/continuous dynamical systems. The separation between continuous and discrete components in sCCP is quite natural. Each sCCP program consists of a set of agents acting on shared variables. Each agent can have different internal states, enabling different actions, while variables describe time-changing quantities. For instance, proteins are described by variables, while genes are modeled as agents, with different internal states depending on which transcription factors are bound to them (see [5] for further details). Therefore, the discrete dynamics of the hybrid automata associated with an sCCP program comes from the different inner states of agents, while the continuous dynamics describes the time evolution of sCCP variables. In [4] it is shown that the hybrid semantics of sCCP is more adherent to the stochastic dynamics than the continuous one.

The purpose of this paper is to extend the method of [4] to stochastic π-calculus. In this case, we do not have any *a-priori* distinction between discrete agents and time-varying variables as in sCCP, hence we need to separate π-agents into two groups, one amenable to continuous approximation and the other inherently discrete. This distinction turns out to be related to conservation properties of the system.

The paper is organized as follows: Section 2 recalls the syntax of a subset of stochastic π-calculus that will be used in the following, while Section 3 briefly presents Hybrid Automata. The identification of discrete components of π-calculus programs is tackled in Section 4, while Section 5 presents the mapping to Hybrid Automata. Conclusions are drawn in Section 6.

2 Stochastic π-calculus

We recall briefly the syntax of stochastic π-calculus processes in Chemical Ground Form (CGF from now on), as defined in [7]. This is a restricted subset of π-calculus (actually, of CCS), which is however sufficient to model biochemical reactions and genetic networks. Essentially, the restriction operator is dropped and no information is passed on channels during a communication. Parameterized communication can however be introduced without substantially changing the language, see [7] for further details.
Processes in CGF are defined by the grammar of Table 1. Essentially, a π-calculus program in CGF consists of a set of reagents (agents/molecules) E and of the

$$\boxed{\begin{array}{c} E ::= \mathbf{0} | X = M, E \\ M ::= \mathbf{0} | \pi.P \oplus M \\ P ::= \mathbf{0} | (X|P) \\ \pi ::= \tau^r | ?x^r | !x^r \\ CGF ::= (E, P) \end{array}}$$

Table 1. Syntax of stochastic π-calculus processes in Chemical Ground Form.

initial solution (network/configuration) P_0.[3] Each reagent is a summation M, whose addends are guarded by actions, either silent (τ^r) or communications on channels (inputs $?x^r$ and outputs $!x^r$). All actions are indexed by a parameter r, which is the rate of an exponential distribution governing their execution time. It is agreed that all occurrences of each channel have the same rate, so that the function ρ, assigning to each distinct channel name and silent action its rate, can be consistently defined.

Remark 1. The same channel name x can be used several times in the definition of reagents in E, so that different agent types can communicate on it. For instance, consider $X_1 = !x.X_1$, $X_2 = ?x.X_2$, $X_3 = !x.X_3$, and $X_4 = ?x.X_4$. x gives rise to 4 possible communications (X_1 with X_2 or X_4 and $X_3$with X_2 or X_4). In general, if there are n occurrences of $!x^r$ and m occurrences of $?x^r$, then there are mn possible interactions on channel x. Actually, an equivalent system can be easily constructed where all possible communications have distinct names: we can replace x with nm new channels $x^r_{i,j}$, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, substituting the i^{th} occurrence of $!x^r.P$ with the sum $!x^r_{i,1}.P \oplus \ldots \oplus !x^r_{i,m}.P$, and the j^{th} occurrence of $?x^r$ with $!x^r_{1,j}.P \oplus \ldots \oplus !x^r_{n,j}.P$. Note that this can cause a quadratic explosion in the description of $CGF(E, P)$, although it leaves the dynamics unchanged.

From now on, we confine ourselves to programs in which each channel appears once as input and once as output. In addition, we index all τ actions with a different integer number, in such a way that τ actions are all distinct.

Example 1. We present here a very simple genetic regulatory network, consisting of one single gene producing a protein that represses its own production. For simplicity, we consider a model in which the gene generates directly the proteins, without the intermediate step of mRNA production. This system can be described by the following set of reactions:

$$\begin{array}{c} G + P \rightarrow_{k_b} G_b + P \\ G_b \rightarrow_{k_u} G \\ G \rightarrow_{k_p} G + P \\ P \rightarrow_{k_d} \emptyset, \end{array} \tag{1}$$

[3] Each solution P can be compactly described by its "marking", i.e. by counting the number of copies of each agent X in parallel in P. This is sufficient to determine the dynamics of the system.

where G is the gene, G_b is the repressed gene and P is the produced protein. Following the conversion rules from chemical reactions prescribed in [7], the π-calculus program in CGF associated to (1) is defined as

$$\begin{aligned} G &= ?b^{k_b}.G_b \oplus \tau_p^{k_p}.(G|P) \\ G_b &= \tau_u^{k_u}.G \\ P &= !b^{k_b}.P \oplus \tau_d^{k_d}.\mathbf{0} \end{aligned} \tag{2}$$

In this encoding, each molecule is represented as a distinct process, and its action capabilities correspond to the different reactions of which the molecule is a reactant. A full program requires also the specification of the initial state of the system, which in this case consists of one single copy of G.

Associating ODE's with π-calculus programs in CGF. Given a π-calculus program (E, P_0) in CGF, we can associate with it a set of ODE's quite straightforwardly. Our presentation differs slightly from [7, 8], in order to simplify the following discussion.
First, we need some preliminary definitions. The function $action(X)$ returns the set of actions of the reactant X; it can be extended to set of agents by $action(\{X_1, X_2\}) = action(X_1) \cup action(X_2)$. For instance, $action(P) = \{b, \tau_d\}$, where P is defined in (2), and $action(\{P, G_b\}) = \{b, \tau_d, \tau_u\}$. Therefore, $action(E)$ is the set of actions of all agents defined in E. In addition, with $\#(X, P)$ ($\#(X, B)$) we denote the number of occurrences of the agent X in the solution P (multiset B). Finally, with $react(\pi)$ and $prod(\pi)$ we indicate the multisets of agents that are consumed and produced by π, respectively. For instance, if $X = !x.(X|X)$ and $Y = ?x.(Y|Y)$, then $react(x) = [\![X, Y]\!]$ and $prod(x) = [\![X, X, Y, Y]\!]$.
The basic idea in associating ODE's with a π-program is to approximate the number of occurrences $\#(X, P)$ of an agent X in the solution P by a continuous quantity, also indicated with X ($\mathbf{X}$, instead, will denote the vector of all variables X).

Definition 1. *Let (E, P_0) be a π-calculus program in CGF, $A \subseteq E$, and $T \subseteq action(E)$.*

1. *The* stoichiometric matrix *$S_{A,T}$ w.r.t. A and T is a $|A| \times |T|$ matrix, with rows indexed by agents of A and columns indexed by actions of T, defined by $S_{A,T}[X, \pi] = \#(X, prod(\pi)) - \#(X, react(\pi))$.*
2. *The rate vector $\phi_{A,T}$ w.r.t. A and T is a $|T|$ vector defined by*[4]

$$\phi_{A,T}[\pi](\mathbf{X}) \begin{cases} 0, & \text{if } react(\pi) \cap A = \emptyset; \\ \rho(\pi)X, & \text{if } react(\pi) \cap A = [\![X]\!]; \\ \rho(\pi)XY, & \text{if } react(\pi) \cap A = [\![X, Y]\!]; \\ \rho(\pi)X(X-1), & \text{if } react(\pi) \cap A = [\![X, X]\!]. \end{cases}$$

[4] The rate function for homodimeric reactions lacks a factor 1/2 w.r.t. the standard definition. This happens because each homodimer $X = !x.X \oplus ?x.X$ counts twice: once for the input $!x$ and once for the output $?x$. This gives a factor two in the rate function canceling out 1/2. Clearly, this must be taken into account while writing models in π-calculus: rates of homodimeric reactions must be halved [7].

3. *The ODE's associated to* (E, P_0) *are* $\dot{\mathbf{X}} = S_{E,action(E)} \cdot \phi_{E,action(E)}(\mathbf{X})$*, with initial conditions given by* $X(0) = \#(X, P_0)$*.*[5]

Consider again the program of Example 1. For $T = action(E)$, we obtain:

$$S_{E,T} = \begin{array}{c} (G) \\ (G_b) \\ (P) \end{array} \begin{pmatrix} -1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{pmatrix} \quad \phi_{E,T} = \begin{pmatrix} \rho(b)GP \\ \rho(\tau_p)G \\ \rho(\tau_d)P \\ \rho(\tau_u)G_b \end{pmatrix} \quad \begin{cases} \dot{G} = \rho(\tau_u)G_b - \rho(b)GP \\ \dot{G}_b = \rho(b)GP - \rho(\tau_u)G_b \\ \dot{P} = \rho(\tau_p)G - \rho(\tau_d)P \end{cases}$$

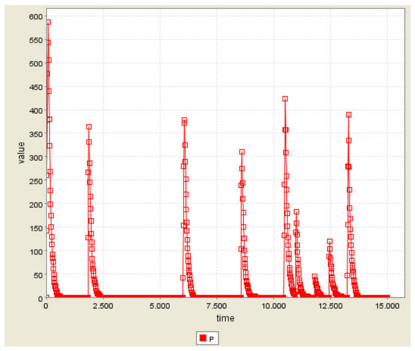
In Figure 1 we compare a stochastic simulation of the π-calculus model (2) with the numerical solution of the associated ODE's. As we can readily see, the two plots are different. In particular, in the stochastic simulation, P is produced in bursts and it follows an irregular oscillatory pattern. The ODE's system, instead, presents a much simpler pattern of evolution, in which the quantity of P converges to an asymptotic value. This divergence is caused by the fact that, approximating continuously the state of the gene, we lose any information on the discrete dynamics of gene's activations and deactivations. As a matter of fact, the same loss occurs when we consider the average trajectory of the stochastic system. In fact, the average presents a trend more similar to that of Figure 1(b) (data not shown), not conveying an adequate description of the dynamics. Consider, for instance, an event triggered when the concentration of P exceeds 100: this event would be activated infinitely often in the stochastic system, but just once in the ODE-based one (or in the average trajectory).

Remark 2. In the mapping from π-calculus to ODE's of Definition 1, the rates of the stochastic processes and of the ODE's are the same, in contrast with the usual praxis in biochemistry, in which rates of ODE's are redefined in terms of concentrations. Indeed, the ODE's that we defined must be thought of as an approximation of the stochastic process, in the sense that they are a first-order approximation of the differential equation for the average of the stochastic system, cf. [1]. This relationship is similar to the one connecting deterministic and stochastic mass action models of chemical reactions, cf. [10].

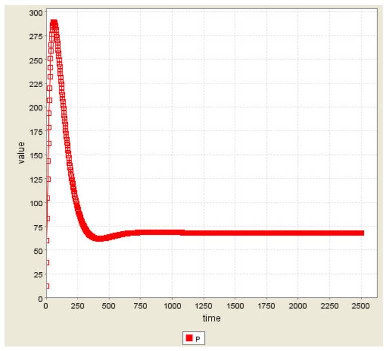
3 Hybrid Automata

In this section we briefly recall the ideas and the definition of hybrid automaton. The reader is referred to [13] for an introductory survey. Hybrid automata are dynamical systems presenting both discrete and continuous evolution. Essentially, they are defined using a set of variables evolving continuously in time, subject to instantaneous changes induced by the happening of discrete *control* events. When discrete events happen the automaton enters its next *mode*, where the laws governing the flow of continuous variables change. Formally, a hybrid automaton is a tuple $H = (V, E, \mathbf{X}, flow, init, inv, jump, reset)$, where:

[5] We could have defined the stoichiometric matrix and the rate vector just for the sets E and $action(E)$, instead of parameterizing them w.r.t. subsets A and T. This parametric version, however, will turn out to be useful in Section 5.



(a) π-calculus model of system (1)



(b) ODE model of system (1)

Fig. 1. 1(a): simulation of the π-calculus model (2). The red line corresponds to P. Parameters of the models are the following: $k_p = 1$, $k_b = 0.0001$, $k_u = 0.0005$, $k_d = 0.01$. The initial configuration consists in a single copy of G. **1(b)**: numerical simulation of ODE's associated to the π-calculus model (2), for the same parameters just given. The evolution of P is different from the stochastic case, as it converges quickly to an asymptotic value. Parameters have been assigned in order to correspond to a situation in which the binding/unbinding dynamics of the repressor P is very slow, when compared to protein production and degradation. Increasing the binding and unbinding strength smooths the behavior of the stochastic system, making it closer to Figure 1(b). We refer the reader to [4] for a more biologically relevant example.

- $\mathbf{X} = \{X_1, \ldots, X_n\}$ is a finite set of real-valued variables (the time derivative of X_j is denoted by $\dot{X}_j$, while the value of X_j after a change of *mode* is indicated by X'_j).
- $G = (V, E)$ is a finite labeled graph, called *control graph*. Vertices $v \in V$ are the *(control) modes*, while edges $e \in E$ are called *(control) switches* and model the happening of a discrete event.
- Associated with each vertex $v \in V$ there is a set of ordinary differential equations[6] $\dot{\mathbf{X}} = \mathit{flow}(v)$ (referred to as the *flow conditions*). Moreover, $\mathit{init}(v)$ and $\mathit{inv}(v)$ are two formulae on $\mathbf{X}$ specifying the *admissible initial conditions* and some *invariant conditions* that must be true during the continuous evolution of variables in v (forcing a change of mode to happen when violated).
- Edges $e \in E$ of the control graph are labeled by $\mathit{jump}(e)$, a formula on $\mathbf{X}$ stating for what values of variables each transition is active (the so called *activation region*), and by $\mathit{reset}(e)$, a formula on $\mathbf{X} \cup \mathbf{X}'$ specifying the change of the variables' values after the transition has taken place.

The *traces* of the system are defined as the time traces of the continuous variables. Notice that the activation conditions are in general non-deterministic (as well as resets), hence there can be different traces starting from the same

[6] Other forms of flow specification are possible (differential inclusions, first order formulae, etc.) but sets of differential equations are sufficient for our purposes here.

initial values. In this paper we are concerned mainly with *simulation* of hybrid automata, i.e. with the generation of a set of admissible traces.

In the following, we will need a product construction for HA, which is almost the classical one [13], the only difference being the treatment of fluxes for variables shared among the factors. In our case, in fact, fluxes are added. Before giving the formal definition of this *flux product*, we put forward some notation. The product $G = G_1 \times G_2$ of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ has vertex set $V_1 \times V_2$ and edges of the form $((v_1, w), (v_2, w))$, where $(v_1, v_2) \in E_1$, or $((v, w_1), (v, w_2))$, where $(w_1, w_2) \in E_2$. Given an edge $e \in E$, the projection $\pi_1(e)$ is defined for all edges $e = ((v_1, w), (v_2, w))$, and is the edge $(v_1, v_2) \in E_1$. Projection π_2 can be defined symmetrically.

Definition 2. *Let* $H_1 = (V_1, E_1, \mathbf{X_1}, flow_1, init_1, inv_1, jump_1, reset_1)$ *and* $H_2 = (V_2, E_2, \mathbf{X_2}, flow_2, init_2, inv_2, jump_2, reset_2)$*. The* flux product *of* H_1 *and* H_2 *is the hybrid automaton* $H_1 \otimes H_2 = (V, E, \mathbf{X}, flow, init, inv, jump, reset)$ *defined by:*

1. $(V, E) = (V_1, E_1) \times (V_2, E_2)$;
2. $\mathbf{X} = \mathbf{X_1} \cup \mathbf{X_2}$;
3. $flow((v_1, v_2)) \restriction_X = flow_1(v_1) \restriction_X + flow_2(v_2) \restriction_X$, *if* $X \in \mathbf{X_1} \cap \mathbf{X_2}$. *Otherwise, if* $X \in \mathbf{X_i}$, *then* $flow((v_1, v_2)) \restriction_X = flow_i(v_i) \restriction_X$;
4. $init((v_1, v_2)) = init_1(v_1) \wedge init_2(v_2)$ *and* $inv((v_1, v_2)) = inv_1(v_1) \wedge inv_2(v_2)$;
5. $jump(e) = jump_1(e_1)$, *if* $e \in E$ *is such that* $\pi_1(e) = e_1$, *otherwise, if* $\pi_2(e) = e_2$, *then* $jump(e) = jump_2(e_2)$;
6. $reset(e) = reset_1(e_1)$ *if* $\pi_1(e) = e_1$, *while* $reset(e) = reset_2(e_2)$ *if* $\pi_2(e) = e_2$.

4 Control Automata

The reactants E of a π-calculus program (E, P_0) in CGF can be broadly separated into two classes, one comprising all those terms which will change in quantity during the evolution (essentially, molecules) and the other containing a collection of terms defining a *control structure*, in such a way that exactly one term of the collection is active in every solution P reachable from P_0. For instance, consider the model of the gene of Example 1. The gene G has two mutually exclusive states: the normal form G and the repressed form G_b. Obviously, we always have a single occurrence of the gene in the system, which can be in one of the two states G and G_b. Essentially, we can think of a gene as a kind of "logical" entity, whose activity depends on its inner state and whose state may change due to interactions with the system.

We want now to find suitable conditions to define collections of terms behaving like the gene in the example, which will be called in the rest of the paper *control automata* (CA), because they can be thought of as automata present in a single copy and synchronizing with the rest of the system. Let R be a CA. First, as the reagents $X_i \in R$ represent different inner states of the CA, there should not be any inner communication between two of them:

$$\forall X \in R, \forall \pi \in action(X), react(\pi) = [\![X, Y]\!] \Rightarrow Y \notin R. \tag{3}$$

In addition, any action involving $X \in R$ (call it an R-state) can change the state, but in any case it cannot destroy nor create more than one R-state in the current solution. Hence R must satisfy:

$$\forall X \in R, \forall \pi \in \mathit{action}(X), \exists! Y \in R : Y \in \mathit{prod}(\pi). \tag{4}$$

A further condition that must be satisfied, is that only actions involving one component of R may change its state:

$$\forall \pi \in \mathit{action}(E), \mathit{react}(\pi) \cap R = \emptyset \Rightarrow \mathit{prod}(\pi) \cap R = \emptyset. \tag{5}$$

The final request on R is that in the initial configuration P_0 of CGF, exactly one state of R is present in P_0 (we indicate $\#(X, P_0)$ with $\#_0(X)$):

$$\#_0(R) = \sum_{X \in R} \#_0(X) = 1. \tag{6}$$

Collecting all the previous conditions together, we are ready to define a control automaton:

Definition 3. *Let (E, P_0) be a π-calculus program in CGF. A subset $R \subseteq E$ is a* control automaton *if and only if it satisfies properties (3), (4), (5), (6).*
A minimal control automaton *R is a CA such that no proper subset of R is a CA. $\mathcal{C}(E)$ denotes the set of all minimal CA of E.*

The conditions satisfied by CA imply that the number of occurrences of elements of R in the system remains constant during each possible computation, in fact equal to one. Formally, we define

Definition 4. *A set $R \subseteq E$ of reagents is* conserved *iff $\#(R, P) = \#(R, P')$, for every pair of solutions P, P' such that $P \xrightarrow{\pi} P'$ (i.e. P can be transformed into P' by the action π), where $\#(R, P) = \sum_{X \in R} \#(X, P)$.*

Theorem 1. *$R \subseteq E$ is a CA $\Rightarrow$ R is conserved.*

Proof. Let P, P' be two configurations such that $P \xrightarrow{\pi} P'$. Clearly, the following relation holds:

$$\#(R, P') = \#(R, P) - \#(R, \mathit{react}(\pi)) + \#(R, \mathit{prod}(\pi)). \tag{7}$$

If $\mathit{react}(\pi) \cap R = \emptyset$, condition (5) implies that $\mathit{prod}(\pi) \cap R = \emptyset$, hence $\#(R, P') = \#(R, P)$ by equation (7). Otherwise, condition (3) implies that $\#(R, \mathit{react}(\pi)) = 1$, while condition (4) states that $\#(R, \mathit{prod}(\pi)) = 1$, hence $\#(R, P') = \#(R, P)$ again by equation (7). ■

Corollary 1. *If $R \subseteq E$ is a CA, then, for each configuration P reachable from P_0, $\#(R, P) = 1$.* ■

The previous theorems state that control automata are indeed entities with different states, one of which only is active at each stage of the evolution of the system: control automata are neither produced nor degraded. This results in a conservation law, in which the sum of the quantity of internal states of the automata always equals one. Conservation laws allow us to use linear algebra to characterize set of conserved states, as we will show in the following.
Before that, we observe that the previous theorem cannot be reversed. Consider this simple system:

$$X = \tau_1.Y \oplus !x.X \quad Y = \tau_2.X \oplus ?x.Y,$$

where X can change state into Y and viceversa, due to a silent action. Moreover, X and Y can synchronize on channel x, a condition violating property (3). Note that all actions τ_1, τ_2, x maintain constant the quantity $\#(X,P)+\#(Y,P)$. If no other action of the system can create or destroy copies of X and Y, clearly $R = \{X, Y\}$ is conserved. R, however, is not a CA, as it fails to satisfy properties (3) and (4). If the starting configuration of the system is such that $\#_0(R) = 1$, then no communication on x will ever be possible, as we will never have a pair of X, Y agents ready to communicate. In this case, we may remove the branch $!x.X$ from X and the branch $?x.Y$ on Y without altering the behavior of the system. The resulting agents X' and Y' are now CA. This justifies the following definition:

Definition 5. *Let $R \subseteq E$. The reduced form $\overline{R}$ of R is obtained by removing all occurrences of channels x such that $react(x) \cap R = \{X, Y\}$.*

We can now prove the following:

Theorem 2. *If $R \subseteq E$ is conserved and $\#_0(R) = 1$, then the reduced form $\overline{R}$ of R is a control automaton.*

Proof. If R is conserved, then for every action $\pi \in \mathcal{A}$

$$\#(R, react(\pi)) = \#(R, prod(\pi)) \tag{8}$$

From this equation, property (5) follows immediately. In addition, property (4) holds for all π such that $\#(R, react(\pi)) = 1$. If no $\pi \in action(E)$ is such that $\#(R, react(\pi)) = 2$, then the set R is a CA. Otherwise, it becomes a CA after removing all $\pi \in action(E)$ is such that $\#(R, react(\pi)) = 2$, i.e. all inner communications in R. The resulting set is exactly the reduced form $\overline{R}$ of R. ■

Theorems 1 and 2 give us a criterion to separate terms belonging to Control Automata (i.e. agents exerting a control activity in the system) from agents whose number changes over time. In fact, we need to identify collections R of reagents that are conserved in the evolution of the system, whose initial quantity is $\#_0(R) = 1$.

Remark 3. The characterization of CA as conserved sets has been suggested to us by the study of conservation properties of Petri Nets [21]. As a matter of fact, it is possible to define a mapping of stochastic π-calculus in CGF to Petri Nets and reason on the latter. However, working directly with π-calculus is more intuitive in this context.

Theorem 3. *Let (E, P_0) be a π-calculus program in CGF. A set R is conserved iff the vector $\mathbf{y}_R$ on E, equal to 1 for $X \in R$ and to 0 otherwise, belongs to the null space of the matrix $A = S^T_{E,action(E)}$.*

Proof. Let $\mathbf{r}$ be a non-negative vector indexed by $action(E)$. If $\mathbf{r}$ represents the set of actions happening in a solution P with multiplicity of each term given by a vector $\mathbf{x}$ on E (i.e. $x_i = \#(X_i, P)$), then the configuration after the happening of $\mathbf{r}$ has multiplicity $\mathbf{x}' = \mathbf{x} + S_{E,action(E)}\mathbf{r}$ (remember that the stoichiometric matrix $S_{E,action(E)}$ gives the net variation of each reagent in E after the happening of each action of $action(E)$). Suppose now $A\mathbf{y}_R = 0$, and $\mathbf{x}$, $\mathbf{x}'$ are such that $\mathbf{x}' = \mathbf{x} + S_{E,action(E)}\mathbf{r}$ for some $\mathbf{r}$. The number of reagents of R in $\mathbf{x}$ is given by $\sum_{\mathbf{y}_R[X]=1} \mathbf{x}[X] = \mathbf{y}_R^T\mathbf{x}$. The net variation of R after $\mathbf{r}$ is

$$\mathbf{y}_R^T\mathbf{x} - \mathbf{y}_R^T\mathbf{x}' = \mathbf{y}_R^T(\mathbf{x} - \mathbf{x}') = \mathbf{y}_R^T(S_{E,action(E)}\mathbf{r}) = (A\mathbf{y}_R)^T\mathbf{r} = 0,$$

hence R is conserved. Viceversa, if R is conserved, then $(A\mathbf{y}_R)^T\mathbf{r} = 0$ for all $\mathbf{r}$, hence $A\mathbf{y}_R = 0$. ■

The previous property gives a clear way to identify Control Automata. All we have to do is find all conserved sets R, i.e. vector of the null space of A whose entries are either zero or one, whose initial value in P_0 is one. Essentially, we have to solve a *zero-one integer programming problem* under the constraints $A\mathbf{y} = 0$ and $\sum_{X\in E} \#_0(X)\mathbf{y}[X] = 1$.
Suppose now we have solved this problem and collected the set of solutions. In this way, we have identified all the candidates to be control automata. Some of them satisfy Definition 4 only after removing inner communications that cannot fire, hence they are CA contingent on the initial conditions. Accepting them as proper CA would make the construction of the next section dependent on initial conditions. To avoid this, we simply remove these pseudo-CA from the solution set. Formally, let $\mathbf{y}_R$ be the 0-1 E-vector that is equal to 1 for agents in R and 0 elsewhere; we consider the set $\mathcal{R}(E) = \{R \subseteq E \mid A \cdot \mathbf{y}_R = 0 \wedge \sum_{X\in R} \#_0(X) = 1\}$ and remove the pseudo-CA obtaining $\mathcal{R}^-(E) = \{R \in \mathcal{R}(E) \mid \forall\pi \in action(E), \#(R, react(\pi)) \leq 1\}$. This new set has some nice closure properties:

Proposition 1. *Let $R, R_1, R_2 \in \mathcal{R}^-(E)$ such that $R_1 \subset R$ and $R_2 \subset R$. Then $R_1 \cap R_2 \neq \emptyset$ and $R_1 \cap R_2 \in \mathcal{R}^-(E)$.* ■

This allows to define consistently the minimal representative $\mu_E(R)$ of a CA $R \in \mathcal{R}^-(E)$ as the intersection of all subsets contained in R:

$$\mu_E(R) = \bigcap_{R'\in\mathcal{R}^-(E), R'\subseteq R} R'.$$

Proposition 1 implies that $\mu_E(R) \in \mathcal{R}^-(E)$, hence the set of *minimal CA* $\mathcal{R}_M(E) = \{\mu_E(R) \mid R \in \mathcal{R}^-(E)\}$ is a subset of $\mathcal{R}^-(E)$. Actually, $\mathcal{R}_M(E)$ is precisely the set we are interested in, according to the following theorem.

Theorem 4. *Let (E, P_0) be in CGF. Then, $\mathcal{C}(E) = \mathcal{R}_M(E)$.* ■

Let's go back to our running example. In this case, the stoichiometric matrix is equal to

$$S_{E,action(E)} = \begin{array}{c} (G) \\ (G_b) \\ (P) \end{array} \begin{pmatrix} -1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 \\ -1 & 1 & -1 & 0 \end{pmatrix}$$

and the initial state of the system consists only of one copy of G. It is immediate to verify that the only solution to $S^T_{E,actionE}\mathbf{y} = \mathbf{0}$ and $\mathbf{y}[G] = 1$ is $\mathbf{y} = (1, 1, 0)$, corresponding to the CA $R = \{G, G_b\}$. This is exactly the set of inner states of the gene, in agreement with the intuitions at the beginning of this section.

Consider now the following π-program: $X = \tau.(X_1|X_2)$, $X_1 =!x.Y_1$, $X_2 = ?x.Y_2$, $Y_1 = \tau_1.X_1$, $Y_2 = \tau_2.X_2$. It holds that $\mathcal{R}_M = \{R_1, R_2\}$, with $R_1 = \{X, X_1, Y_1\}$ and $R_2 = \{X, X_2, Y_2\}$. However, these sets are not disjoint, in contrast with the intuition of control automata as distinct objects. Actually, R_1 and R_2 can be "separated" simply replacing X with $X^1_0|X^2_0$, $X^1_0 =!z.X_1$ and $X^2_0 =?z.X_2$, where $\rho(z) = \rho(\tau)$, an operation not altering the dynamics. Although now $R_1 = \{X^1_0, X_1, Y_1\}$ and $R_2 = \{X^2_0, X_2, Y_2\}$ are disjoint, their evolution is still entangled, as they need to synchronize on z to evolve. In order to keep the mapping to HA simple, we suppose in the following that the set $\mathcal{C}(E)$ of minimal CA of E satisfy the following two conditions:

Definition 6. *Let (E, P_0) be a π-calculus program in CGF with control automata $\mathcal{C}(E)$.*

1. *$\mathcal{C}(E)$ is* independent *iff for all $R_1, R_2 \in \mathcal{C}(E)$, $R_1 \cap R_2 = \emptyset$.*
2. *$\mathcal{C}(E)$ is* non-interacting *iff no two CA of $\mathcal{C}(E)$ can synchronize on an action $\pi \in action(E)$, i.e. there does not exist $R_1, R_2 \in \mathcal{C}(E)$, $\pi \in action(E)$ such that $react(\pi) \cap R_1 \neq \emptyset \wedge react(\pi) \cap R_2 \neq \emptyset$.*

Remark 4. The problem of finding if a 0-1 integer programming problem has a feasible solution is known to be $\mathcal{NP}$-complete [9]. This makes the use of the method defined above potentially troublesome in terms of computational cost. In addition, there is no a-priori bound on the size of the set of solutions $\mathcal{R}(E)$ that needs to be processed to obtain $\mathcal{R}_M(E)$. Therefore, we need to study the complexity of the problem of determining the set of minimal CA, giving, if possible, more refined algorithms. We leave further investigation of these issues for the future.

5 From π-calculus to Hybrid Automata

In order to map a π-calculus program (E, P_0) into CGF to a Hybrid Automaton (HA), we process independently each minimal Control Automaton R of (E, P_0), associating a HA to R. Then, these HA are combined together using the flux product construction of Definition 2. In the following, we suppose that (E, P_0) has independent and non-interacting CA (cf. Definition 6).

Before giving the details of the mapping, we need to fix some preliminary notation. Let $R \in \mathcal{C}(E)$ be a CA of E. The set of *discrete actions of* R is $T_d(R) = \{\pi \in action(E) \mid react(\pi) \cap R \neq \emptyset \wedge react(\pi) \cap prod(\pi) \cap R = \emptyset\}$, i.e. the set of actions changing the state of R. Similarly, the set of *continuous actions of* R is $T_c(R) = \{\pi \in action(E) \mid react(\pi) \cap prod(\pi) \cap R \neq \emptyset\}$, i.e. the set of actions looping on a state of R. Given an action $\pi \in T_d(R) \cup T_c(R)$, its starting state is $start(\pi, R) = X$ iff $react(\pi) \cap R = \{X\}$. Similarly, if $prod(\pi) \cap R = \{Y\}$, then the ending state of π is $end(\pi, R) = Y$. With $T_d(R, X) = \{\pi \in T_d(R) \mid start(\pi, R) = X\}$ we indicate the subset of discrete transitions of R starting at X; a similar definition applies to $T_c(R, X)$. The set T_C of *uncontrolled continuous actions* is $T_C = action(E) \setminus \bigcup_{R \in \mathcal{R}_M(E)} (T_d(R) \cup T_c(R))$. Finally, the *continuous terms* of (E, P_0) are those not belonging to a CA, i.e. $Cont(E) = \{X \in E \mid X \notin \bigcup_{R \in \mathcal{C}(E)} R\}$.

Consider now a CA $R \in \mathcal{C}(E)$. The HA associated to R will be $H(R) = (V(R), E(R), \mathbf{X}, flow, init, inv, jump, reset)$. We discuss how to define its components separately.

Control Graph. The discrete modes $V(R)$ of the HA are simply the different elements of the CA R, i.e. $V(R) = \{\sigma_X \mid X \in R\}$. The edges of the control graph are those implied by the discrete transitions $T_d(R)$: for $\pi \in T_d(R)$, we add the edge $(start(\pi, R), end(\pi, R))$.

Continuous Flow. The (continuous) variables $\mathbf{X}$ of the system are one for each term in $Cont(E)$. These variables, in a state $\sigma_X \in V(R)$, are subject only to the effect of continuous transitions $T_c(R, X)$. The idea to define their continuous flow is simply to restrict the method of Definition 1 to transitions $T_c(R, X)$ for each X in R. Essentially, we define the local stoichiometric matrix $S_{Cont(E), T_c(R,X)}$ and the local rate vector $\phi_{Cont(E), T_c(R,X)}$ according to Definition 1, and we set $flow(\sigma_X) = S_{Cont(E), T_c(R,X)} \cdot \phi_{Cont(E), T_c(R,X)}(\mathbf{X})$. Note that, according to Definition 1, $\phi_{Cont(E), T_c(R,X)}$ depends only on the variables in $Cont(E)$.
The invariant conditions are not needed in our mapping, hence $inv(\sigma_X) = true$, while initial conditions are defined according to the initial configuration P_0 of (E, P_0): if $\#_0(X) = 0$, then $init(\sigma_X) := false$, while if $\#_0(X) = 1$ then $init(\sigma_X) := \bigwedge_{Y \in Cont(E)} (Y = \#_0(Y))$.

Discrete Transitions. The most delicate part of the mapping is the definition of the activation conditions and of the resets of discrete transitions. The point is that in the stochastic process associated to (E, P_0), transitions of $T_d(R)$ are stochastic, with execution time exponentially distributed according to their rate. This stochastic duration needs to be removed taking into account the *timing* of the corresponding event. The simplest possibility is that of *firing the transition at its expected time.* There is however a complication given by the fact that the rate of the transition can depend on some terms that are approximated as continuous. The associated stochastic process is, therefore, a *non-homogeneous Poisson process* [20] with time-dependent rate $\phi_{Cont(E), action(E)}[\pi](\mathbf{X}(t)) = \phi_\pi(\mathbf{X}(t))$. In

this case, given the *cumulative rate* at time t, $\Lambda(t) = \int_0^t \phi_\pi(\mathbf{X}(s))ds$, the probability of firing within time t is $F(t) = 1 - e^{-\Lambda(t)}$. The expected value of $\Lambda(T)$ at the time T of firing is $E[\Lambda(T)] = 1$ (cf. [4, 20]). Hence, we can fire the transition whenever $\Lambda(t) \geq 1$. In order to describe this condition within HA framework, we introduce a new (clock) variable Z_π, with initial value 0, evolving according to the equation $\dot{Z}_\pi = \phi_\pi(\mathbf{X})$. When the transition fires, we reset all variables Z_π to zero, due to memoryless property of CTMCs.
The jump predicate can thus be defined as follows. If $\pi \in T_d(R, X)$ is such that $react(\pi) = \{X\}$, then $jump(\pi) := Z_\pi \geq 1$. Else, if $react(\pi) = \{X, Y\}$, then $jump(\pi) := Z_\pi \geq 1 \wedge Y \geq 1$. The fact that CA are non-interacting guarantees that $Y \in Cont(E)$. Moreover, each discrete transition will be *urgent*, meaning that it will fire as soon as its guard becomes true.
Resets of discrete transitions need to take into account the modifications induced on continuous variables. Moreover, they need to set to zero variables Z_π. Hence, for $\pi \in T_d(R, X)$, we set $reset(\pi) := \bigwedge_{Y \in Cont(E)}(Y' = Y + \#(Y, prod(\pi)) - \#(Y, react(\pi))) \wedge \bigwedge_{\pi_1 \in T_d(R)}(Z'_{\pi_1} = 0)$.

We collect now the previous discussion into a formal definition.

Definition 7. *Let $R \in \mathcal{C}(E)$ be a CA of a π-calculus program (E, P_0) in CGF. $H(R) = (V(R), E(R), \mathbf{W}, flow, init, inv, jump, reset)$, the Hybrid Automaton associated to R, is defined by:*

1. *$V(R) = \{\sigma_X \mid X \in R\}$ and $E(R) = \{(start(\pi, R), end(\pi, R)) \mid \pi \in T_d(R)\}$;*
2. *$\mathbf{W} = \mathbf{X} \cup \mathbf{Z}$, where $\mathbf{X}$ is the vector of variables on $Cont(E)$ and $\mathbf{Z} = \{Z_\pi \mid \pi \in T_d(R)\}$;*
3. *for each $\sigma_X \in V(R)$, $flow(\sigma_X)[Y] := S_{Cont(E),T_c(R,X)}[Y, \cdot]\phi_{Cont(E),T_c(R,X)}$ for $Y \in \mathbf{X}$ and $flow(\sigma_X)[Z_\pi] := \phi_{Cont(E),action(E)}[\pi]$ for $Z_\pi \in \mathbf{Z}$;*
4. *for each $\sigma_X \in V(R)$, $inv(\sigma_X) := true$;*
5. *for each $\sigma_X \in V(R)$, $init(\sigma_X) := false$ if $\#_0(X) = 0$ and $init(\sigma_X) := \bigwedge_{Y \in Cont(E)}(Y = \#_0(Y)) \wedge \bigwedge_{\pi \in T_d(R)}(Z_\pi = 0)$ if $\#_0(X) = 1$;*
6. *for each $\pi \in T_d(R)$, $jump(\pi) := (Z_\pi \geq 1) \wedge \bigwedge_{Y \in react(\pi) \cap Cont(E)}(Y \geq 1)$;*
7. *for each $\pi \in T_d(R)$, $reset(\pi) := \bigwedge_{Y \in Cont(E)}(Y' = Y + \#(Y, prod(\pi)) - \#(Y, react(\pi))) \wedge \bigwedge_{\pi_1 \in T_d(R)}(Z'_{\pi_1} = 0)$.*

The previous definition gives us the recipe to associate an hybrid automaton to each control automaton of a π-calculus program. However, there are some actions that are not taken into account up to now, namely those belonging to the set of uncontrolled continuous actions T_C. In order to take into account their effect, we define a special hybrid automaton with one single state.

Definition 8. *Let (E, P_0) be a π-calculus program in CGF and T_C the set of uncontrolled continuous transitions. The Hybrid Automaton $H(T_C)$ associated to T_C is $H(T_C) = (V, E, \mathbf{X}, flow, init, inv, jump, reset)$, where $V = \{\sigma\}$, $E = \emptyset$, $jump := false$, $reset := false$, $inv(\sigma) := true$, $init(\sigma) := \bigwedge_{X \in Cont(E)} X = \#_0(X)$, and $flow(\sigma) := S_{Cont(E),T_C}\phi_{Cont(E),T_C}(\mathbf{X})$, where $\mathbf{X}$ is the vector of variables on $Cont(E)$.*

We are finally ready to define the HA associated to the whole π-program: we simply need to take the flux product of the HA coming from the different CA of $\mathcal{C}(E)$ and of the HA coming from uncontrolled actions.

Definition 9. *Let (E, P_0) be a π-calculus program in CGF with a non-interacting and independent $\mathcal{C}(E)$ of CA. The Hybrid Automaton $H(E)$ associated to (E, P_0) is*

$$H(E) = H(T_C) \otimes \bigotimes_{R \in \mathcal{C}(E)} H(R).$$

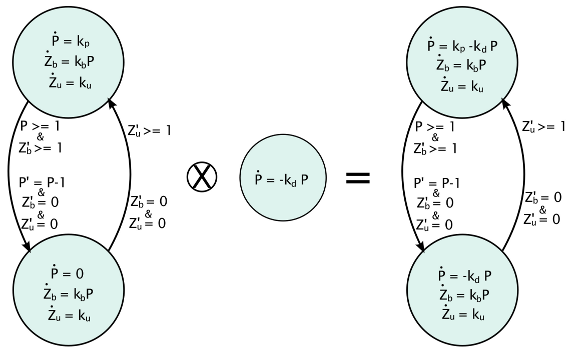


Fig. 2. Hybrid Automata obtained for the π-calculus process of Example 1.

Consider again the simple autoregulatory gene network of Example 1. According to the computation at the end of Section 4, the system has one control automata, namely $R = \{G, G_b\}$, and one continuous term P. The Hybrid Automata $H(R)$ and $H(T_C)$ are shown in Figure 2, together with their product $H(E)$. We can see that the final HA has three continuous variables, one corresponding to P and the other two, Z_b, Z_{τ_u}, associated to the discrete edges $T_d(R) = \{b, \tau_u\}$. We can also see how the final vector field for P is the sum of the vector fields for P of the two factors. In Figure 3, we show a simulation of $H(E)$ for the same parameters as in Figure 1. As we can see, the dynamics of the hybrid automaton resembles very closely the behavior of the stochastic system rather than the one of the ODE's, preserving alternating spikes. Hence, keeping part of the discreteness of the original system was enough to maintain qualitatively the oscillatory pattern of Figure 1(a). We stress the fact that the correspondence between the plots of Figures 1(a) and 3 is only partially quantitative. For instance, in both the stochastic and the hybrid model P overcomes the threshold of 100 infinitely often. As a matter of fact, the noise of the stochastic systems perturbs dramatically the period of oscillations, which is highly regular in the hybrid system, due to the urgent policy for transitions. In [4] we suggested the use of non-determinism to reintroduce a certain degree of variability.

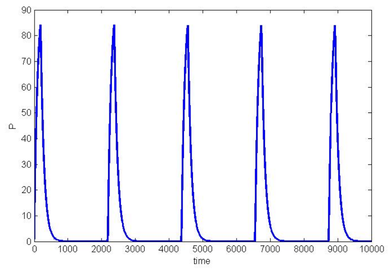


Fig. 3. Simulation of the Hybrid Automaton of Figure 2, for the same parameters as in Figure 1.

Remark 5. In the definition of the mapping, we assumed that $\mathcal{C}(E)$ is independent and non-interacting. The choice of independence, i.e. of having disjoint minimal CA, is fundamental in order to use the flux product construction. Non-interactivity, instead, can be dropped by introducing a product of HA that allows synchronization of discrete transitions. However, we did not follow this direction here, to keep the presentation simpler.

6 Conclusions

In this paper we provided a restricted form of stochastic π-calculus, the so called Chemical Ground Form [7], with a semantics based on hybrid automata. The most difficult aspect is the identification, without the availability of external knowledge, of portions of a π-calculus program acting as discrete controlling components. This is necessary to define the discrete skeleton of the HA and it is related to conservation laws of the system.
We argued that the hybrid semantics is more appropriate to reproduce qualitatively the stochastic behavior of the simple genetic network of Example 1. The same observation seems to apply to a broad class of genetic circuits, cf. [4] for further details.
Comparing qualitatively the dynamical evolution of systems described with different mathematical formalisms is itself an intriguing problem. We believe that a reasonable choice can be based on a temporal logic to describe dynamical properties, thus equating behavioral equivalence with equi-satisfiability.

As a matter of fact, Hybrid Automata may be used also to tackle the discreteness given by molecules present in small quantities in the system. The idea is roughly that of separating actions into two groups, those amenable of continuous approximation and those to be kept discrete. Different partitions correspond to different HA, *de facto* associating to each π-calculus model a lattice of HA, differing in the degree of discreteness. Dynamic partition schemes can be considered as well. This approach will result in a flexible hybrid semantics, capable of dealing effectively with size effects and multi-scale systems.
Another important issue consists in stating a clear connection between our hybrid semantics and standard models of biochemical networks, like the chemical

master equation (CME) or mass action ODEs. A possibility in this sense is to manipulate the CME similarly to [12], where authors formally justify hybrid simulation schemes mixing stochastic differential equations with discrete jump Markov processes. However, the advantage of our hybrid semantics with respect to hybrid simulation strategies like [17, 12] is that HA offer a powerful and flexible framework both for simulation and for verification, computationally more efficient than stochastic processes.

References

1. L. Bortolussi. A master equation approach to differential approximations of stochastic concurrent constraint programming. In *Proceedings of QAPL'08.*, 2008.
2. L. Bortolussi and A. Policriti. Dynamical systems and stochastic programming I - ordinary differential equations. *Submitted to Trans. of Comp. Sys. Bio.*, 2008.
3. L. Bortolussi and A. Policriti. Stochastic concurrent constraint programming and differential equations. In *Proceedings of QAPL'07*, ENTCS volume 16713, 2007.
4. L. Bortolussi and A. Policriti. Hybrid approximation of stochastic concurrent constraint programming. In *proceedings of IFAC'08.*, 2008.
5. L. Bortolussi and A. Policriti. Modeling biological systems in concurrent constraint programming. *Constraints*, 13(1), 2008.
6. M. Calder, S. Gilmore, and J. Hillston. Modelling the influence of RKIP on the ERK signalling pathway using the stochastic process algebra PEPA. *Trans. of Comp. Sys. Bio.*, 4230:1–23, 2006.
7. L. Cardelli. From processes to ODEs by chemistry. *downloadable from* `http://lucacardelli.name/`, 2006.
8. L. Cardelli. On process rate semantics. *TCS*, 2007.
9. M.R. Garey and D. S. Johnson. *Computers and Intractability: a Guide to the Therory of NP-Completeness.* Freeman, 1979.
10. D. Gillespie. The Chemical Langevin Equation. *Jo. of Chem. Phys.*, 113(1):297–306, 2000.
11. D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. of Phys. Chem.*, 81(25), 1977.
12. E.L. Haseltine and J.B. Rawlings. On the origins of approximations for stochastic chemical kinetics. *J. Chem. Phys.*, 123, 2005.
13. T. A. Henzinger. The theory of hybrid automata. In *proceedings of LICS'96*, 1996.
14. J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.
15. J. Hillston. Fluid flow approximation of PEPA models. In *proceedings of QEST'05*, 2005.
16. H. Kitano. Computational systems biology. *Nature*, 420:206–210, 2002.
17. N. A. Neogi. Dynamic partitioning of large discrete event biological systems for hybrid simulation and analysis. In *proceedings of HSCC'04*, volume 2993 of *LNCS*, pages 463–476, 2004.
18. C. Priami and P. Quaglia. Modelling the dynamics of biosystems. *Briefings in Bioinformatics*, 5(3):259–269, 2004.
19. A. Regev and E. Shapiro. Cellular abstractions: Cells as computation. *Nature*, 419, 2002.
20. S. M. Ross. *Stochastic Processes.* Wiley, New York, 1996.
21. D. J. Wilkinson. *Stochastic Modelling for Systems Biology.* Chapman & Hall, 2006.