Hybrid approximation of stochastic process algebras for systems biology

Luca Bortolussi * **Alberto Policriti** **

* *Dept. of Mathematics and Computer Science, University of Trieste, CBM, Area Science Park, Trieste, Italy. (e-mail: luca@ dmi.units.it).*
** *Dept. of Mathematics and Computer Science, University of Udine, Istituto di Genomica Applicata, Udine, Italy (e-mail: alberto.policriti@ dimi.uniud.it)*

Abstract: We present a technique to approximate models of biological systems written in a "distilled" version of stochastic Concurrent Constraint Programming (sCCP), a stochastic programming methodology based on logic programming. Our technique automatically associates to a stochastic model an hybrid automaton, i.e. a dynamical system where continuous and discrete dynamics coexist. The hybrid automata generated in this way are, in certain cases, capable of capturing aspects of the dynamics of stochastic processes that are lost in approximations based solely on ordinary differential equations. In particular, they work better for those systems whose sCCP model contains explicit *logical* mechanisms of control. In the paper we outline the general technique to perform this association and we discuss some issues related to its applicability.

Keywords: Stochastic programming, Hybrid systems, Approximate analysis, Computer simulation, Biosystems

1. INTRODUCTION

Mathematical modeling in Computational Systems Biology is dominated by two formalisms, the first one being (mainly ordinary) differential equations and the second one being continuous-time stochastic processes [Kitano, 2001, 2002]. Both methodologies have their roots reaching back to physical and chemical arguments that, at least for modeling biochemical reactions, give strong foundations to the approach. Recently, stochastic modeling has received a lot of attention, thanks also to the use of process algebras (in their stochastic variants) as modeling languages that allow to construct models in a compositional and reusable way [Priami and Quaglia, 2004, Regev and Shapiro, 2002].

It is well known that the choice of "mathematics" to use is not just a matter of personal taste, but rather it is connected with the specific features of the system under analysis (see Wilkinson [2006]). Differential models are appealing, as they count on a (mature) bulk of mathematical instruments for their analysis, numerical solvers above all. However, biological processes often have inherently discrete components (they are ultimately constituted of molecules interacting) and hence a continuous approximation may lead to incorrect results, especially when small-sized populations are taken into account. Stochastic models can be more precise, as they are intrinsically discrete and also able to capture the noise effects. However, the exact simulation of discrete/stochastic models with the (celebrated) Gillespie algorithm [Gillespie, 1977] is computationally more expensive and, in general, the analysis is more difficult.

Relating stochastic process algebras (SPA) and differential equation models is a difficult and interesting problem. Specifically, we are interested in finding methods that automatically translate SPA models into ODEs. This translation must somehow preserve both the structure and the behavior of the models: if we start from a SPA program, the associated set of ODEs should be *behaviorally equivalent* to it, in the sense of showing the same (at least qualitative) dynamics. Some techniques have been proposed in the literature to cope with this problem [Hillston, 2005, Cardelli, 2006, Bortolussi and Policriti, 2007], working well for some systems but failing to preserve behavior in general. A paradigmatic example is given in Section 3 and the full problem can be classified as still open.

In this paper we focus on translating stochastic programs written in a simplified version of sCCP. Hence, we will start our trip by presenting its syntax (Section 2.1) and the translation procedure to ODEs (Bortolussi [2007], Bortolussi and Policriti [2007], Section 2.2). Then, we will deal with an example where this translation *fails* to preserve the dynamic behavior, i.e. the Repressilator [Elowitz and Leibler, 2000], a synthetic genetic circuit that exhibits oscillatory behavior (Section 3): We will start from a simple stochastic model that manifest neat oscillations, while the solution of the associated ODEs does not oscillate at all. Inspecting the transformation procedure, we will note how, approximating a boolean state of our processes with a continuous variable, we are in fact making a questionable assumption. Next we show how, by avoiding this approx-

⋆ This work was partly supported by FIRB-LIBi and PRIN "BISCA".

imation, go straight to a decoupling of a finite number of possible scenarios and, thereafter, to *hybrid automata*.

Hybrid automata (HA) are a formalism mixing discrete and continuous ingredients: essentially, they are finite automata extended with a set of variables evolving continuously in each state, according to state-specific differential equations. The discrete dynamics is given by transitions between states, which are triggered when activation conditions on variables are satisfied and, when taking place, also reset the value of some variables. We recall the basics of hybrid automata in Section 2.3.

The heart of the paper is Section 4, where the translation from sCCP to hybrid automata is presented, analyzed, and exemplified. The basic idea is to identify a finite set of "states" of the stochastic system, each characterized by a specific dynamics. These "states" will constitute the discrete skeleton of the hybrid automaton, while continuous laws and discrete transitions will be defined according to the dynamics within each "state". In particular, we will show that the hybrid version of the Repressilator (our running example) preserves the oscillatory behavior (its main behavioral property). These preliminary results and the flexibility offered by the interplay between discrete and continuous dynamics suggest HAs an interesting target formalism for the approximation of stochastic process algebras and, perhaps, for more general modeling techniques with a stochastic ingredient.

The translation defined here certainly does not solve all the problems. Most notably, there is still a qualitative divergence between stochastic models and the corresponding hybrid automata when the concentration of species is low. We comment on this problem in the final section, sketching a brief classification of sources of non-equivalence between discrete stochastic models and their continuous approximations. We conclude outlining an alternative use of the discrete ingredient—and especially of non-determinism—as a possible way to improve our technique.

2. PRELIMINARIES

In this section we briefly recall the basic notions we need for the following discussion. We start presenting a simplified version of Stochastic Concurrent Constraint Programming (sCCP, Section 2.1), then we show a method for associating ordinary differential equations to sCCP programs (Section 2.2). Finally, we recall the definition of hybrid automata (Section 2.3).

2.1 Stochastic Concurrent Constraint Programming

Distilled stochastic Concurrent Constraint Programming ((d)sCCP) is a simplified version of sCCP [Bortolussi, 2006], a stochastic extension of CCP [Saraswat, 1993].

A (d)sCCP program consists of a set of agents interacting via a *shared store*, containing a finite set of variables $\mathbf{X} = \{X_1, \ldots, X_n\}$, usually taking integer values.[1] A configuration $\mathbf{c}$ of the store is simply a valuation of the variables $\mathbf{X}$. The basic action π executable by agents is a *guarded update* of some variables, of the form $g(\mathbf{X}) \to u(\mathbf{X}, \mathbf{X}')$, where $g(\mathbf{X})$ is an inequality predicate on variables $\mathbf{X}$ and

[1] More structured numerical domains are possible.

$Prog = D.N$	$D = \varepsilon \mid D.D \mid p : -A$
$\pi = [g(\mathbf{X}) \to u(\mathbf{X}, \mathbf{X}')]_\lambda$	$M = \pi.G \mid M + M$
$G = \mathbf{0} \mid p \mid M$	$A = \mathbf{0} \mid M$
$N = A \mid A \parallel N$	

Table 1. Syntax of restricted sCCP.

$u(\mathbf{X}, \mathbf{X}')$ is a predicate on $\mathbf{X}, \mathbf{X}'$ of the form $\mathbf{X}' = \mathbf{X} + \mathbf{k}$ ($\mathbf{X}'$ denotes variables of $\mathbf{X}$ after the update), with $\mathbf{k}$ a vector of constants.[2] In addition, the language has all the basic constructs of process algebras: non-deterministic choice, parallel composition, and recursive calls. The characteristic feature of (d)sCCP is the fact that each action π is given a *stochastic duration* by associating to it an exponentially distributed random variable, whose rate depends on the state of the system through a function $\lambda : \mathbf{X} \to \mathbb{R}^+$. Stochastic actions are denoted by $[\pi]_\lambda$.

Definition 1. A (d)sCCP program (or (d)sCCP-network) is a tuple $\mathcal{N} = (Prog, \mathbf{X}, init(\mathbf{X}))$, where

(1) *Prog* is defined according to the grammar of Table 1;
(2) $\mathbf{X}$ is the set of variables of the store (with global scope);
(3) $init(\mathbf{X})$ is a predicate on $\mathbf{X}$ of the form $\mathbf{X} = \mathbf{x_0}$, assigning an *initial value* to store variables.

Notably, in Table 1 the use of parallel operator is restricted to the top level of the network: a fact soon implying the lemma below.

Definition 2. A (d)sCCP agent is *sequential* if it does not contain any occurrence of the parallel operator $\parallel$.

Lemma 1. The initial configuration N of a (d)sCCP network $\mathcal{N}$ is the parallel composition of sequential agents, $N = A_1 \parallel \ldots \parallel A_n$, called *components*. In addition, the number of components is *constant* at run-time.

The structural operational semantic of the language [Bortolussi, 2006] is given by a transition relation, from which a Continuous Time Markov Chain [Norris, 1997] can be inferred.

Each sequential (d)sCCP agent can be conveniently represented as a graph, with vertices corresponding to different stochastic choices and edges corresponding to actions. The edges are labeled by the guard, the update, and the rate function of the corresponding action.
Such graphs can be constructed from the syntactic tree of the sequential agent, simply replacing a node corresponding to the call of a procedure p with the syntactic tree of p. This needs to be done at most once for each procedure p; all other nodes corresponding to calls to p are removed and their incoming edge redirected to the root of the unique copy of the syntactic tree of p.
The resulting graph for each component is called the *reduced transition system (RTS)*:

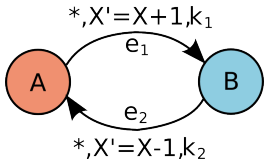
Definition 3. The RTS for a sequential (d)sCCP agent A, acting on variables $\mathbf{X}$, is a tuple $RTS(A) = (S(A), E(A), guard, update, rate)$, where:

[2] The notion of entailment of CCP (see Saraswat [1993]) takes here a very simple form: a guard $g(\mathbf{X})$ is entailed by a configuration $\mathbf{c}$, $\mathbf{c} \vdash g(\mathbf{X})$, if $g(\mathbf{c})$ is true.

(1) $(S(A), E(A))$ is a directed graph with vertices $S(A)$ (also called RTS-states) and edges $E(A)$;
(2) each RTS-state $\sigma \in S(A)$ corresponds to a stochastic choice of A;
(3) each edge $e \in E(A)$ corresponds to a different basic action $[g(\mathbf{X}) \to u(\mathbf{X}, \mathbf{X}')]_{\lambda(\mathbf{X})}$ executable by A;
(4) the functions *guard*, *update*, and *rate* label edges $e \in E(A)$. Specifically, $guard(e) = g(\mathbf{X})$, $update(e) = u(\mathbf{X}, \mathbf{X}')$, and $rate(e) = \lambda(\mathbf{X})$.

As an example, below is a (d)sCCP process composed by one single component A, together with its RTS ($*$ is shorthand for *true*):

A :- $[* \to X' = X + 1]_{k_1}$.B
B :- $[* \to X' = X - 1]_{k_2}$.A



Definition 1 and Lemma 1 easily imply the following:

Theorem 1. The RTS of a (d)sCCP sequential component is always a finite graph (proof can be found in Bortolussi [2007]).

In [Bortolussi, 2007] and [Bortolussi and Policriti, 2006], we argued that sCCP can be conveniently used for modeling a wide range of biological systems, like biochemical reactions, genetic regulatory networks, the formation of protein complexes, and the process of folding of a protein. Actually, (d)sCCP suffices to deal with the first two classes of systems.

2.2 sCCP to ODE

In order to simplify the analysis of an sCCP model, we can define a fluid-flow approximation of the system, by treating variables as continuous and describing their time-evolution by means of ODEs [Bortolussi, 2007, Bortolussi and Policriti, 2007].

Starting from a (d)sCCP network $\mathcal{N}$, with initial configuration $N = A_1 \parallel \ldots \parallel A_n$, we build the RTS for each sequential component A_i. Then, letting $S(N) = S(A_1) \cup \ldots \cup S(A_n)$ and $E(N) = E(A_1) \cup \ldots \cup E(A_n)$, we associate a continuous variable to each RTS-state of $S(N)$. Such variables will be governed by the differential equations, like all variables $\mathbf{X}$ of the store.

Next, we build what we will call the *interaction matrix* I of our sCCP-network. The interaction matrix will have as many rows as system's variables (store variables and RTS-state variables) and as many columns as the edges in $E(N)$. In this matrix we store the updates (constant by definition) that each transition induces on stream variables, putting also a $^{+}\!/_{-}1$ in correspondence to the enter/exit RTS-states of the edges. To write the ODEs, we simply need to store in a vector r the (functional) rates of each transition of the RTS, following the same order used in the interaction matrix. In addition, we multiply such rates by the indicator function of the guards of the edges and by the variables corresponding to the exit RTS-state of the transition. The vector *ode* of ODEs is simply obtained as

$$ode = I \cdot r.$$

For the previous example, we have:

$$I = \begin{array}{c|cc} & e_1 & e_2 \\ \hline X & 1 & -1 \\ A & -1 & 1 \\ B & 1 & -1 \end{array} \qquad r = \begin{pmatrix} k_1 \cdot A \\ k_2 \cdot B \end{pmatrix} \qquad ode \begin{cases} \dot{X} = k_1 \cdot A - k_2 \cdot B \\ \dot{A} = -k_1 \cdot A + k_2 \cdot B \\ \dot{B} = k_1 \cdot A - k_2 \cdot B \end{cases}$$

2.3 Hybrid automata

In this section we briefly recall the ideas and the definition of hybrid automaton. We will not be detailed, as we refer the reader to [Henzinger, 1996] for an introductory survey. Hybrid automata are dynamical systems presenting both discrete and continuous evolution. Essentially, there is a set of variables evolving continuously in time, subject to abrupt changes induced by the happening of discrete (instantaneous) *control* events. When discrete events happen the automaton enters its next *mode* (its next *state* in the finite automata jargon), 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:

- $\mathbf{X} = \{X_1, \ldots, X_n\}$ 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 [3] $\dot{\mathbf{X}} = flow(v)$ (referred to as the *flow conditions*). Moreover, $init(v)$ and $inv(v)$ are two predicates 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 $jump(e)$, a predicate on $\mathbf{X}$ stating for what values of variables each transition is active (the so called *activation region*), and by $reset(e)$, a predicate 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 essentially 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 initial values.
In this paper we are not concerned with model checking issues, but rather with *simulation* of hybrid automata, i.e. with the generation of a set of admissible traces.

3. A PARADIGMATIC EXAMPLE: THE REPRESSILATOR

The *Repressilator* [Elowitz and Leibler, 2000] is an artificial biochemical clock composed of three genes expressing three different proteins, **tetR**, **λcI**, **LacI**, exerting

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

$$\begin{aligned}\mathrm{Neg}(X,R) :\!\!- \quad & [* \to X' = X+1]_{k_p}.\mathrm{Neg}(X,R) \\ + \; & [R \geq 1 \to *]_{k_b R}.[* \to *]_{k_u}.\mathrm{Neg}(X,R)\end{aligned}$$

$$\mathrm{Degrade}(X) :\!\!- [X > 0 \to X' = X-1]_{k_d X}.\mathrm{Degrade}(X)$$

$$\mathrm{Neg}(A,C) \parallel \mathrm{Neg}(B,A) \parallel \mathrm{Neg}(C,B) \parallel \mathrm{Degrade}(A) \parallel \mathrm{Degrade}(B) \parallel \mathrm{Degrade}(C)$$

Table 2. sCCP code for the Repressilator. We are using three template processes (with template variables X for the protein and R for the repressor) that are instantiated with the three global stream variables of the system, namely A, B, C.

a cyclical inhibitory function on each other's gene expression. The expected behavior is an oscillation of the concentrations of the three proteins and a non-stochastic automata-based simulation can be found in Antoniotti et al. [2003]. An abstract stochastic model of Repressilator can be found in Blossey et al. [2006], where the authors propose a general formalism based on π-calculus to model genetic regulatory networks. In such a model, each gene is represented as a gate with one output (the expressed protein) and different inputs, corresponding to the different regulatory mechanisms it is subject to. In particular, negative gates model genes whose expression can be inhibited by a repressor; more specifically, inhibition is obtained as a stochastic delay. The (d)sCCP code for *NEG* gates can be found in Table 2. In this formalism, the Repressilator can be described by the combination of three *NEG* gates, in addition to the degradation mechanisms for the three regulatory proteins. The resulting (d)sCCP model is shown in Table 2. In Figure 1(a) we show a trace of the stochastic model generated by a simulator of (d)sCCP based on Gillespie algorithm. The oscillatory behavior is manifest.

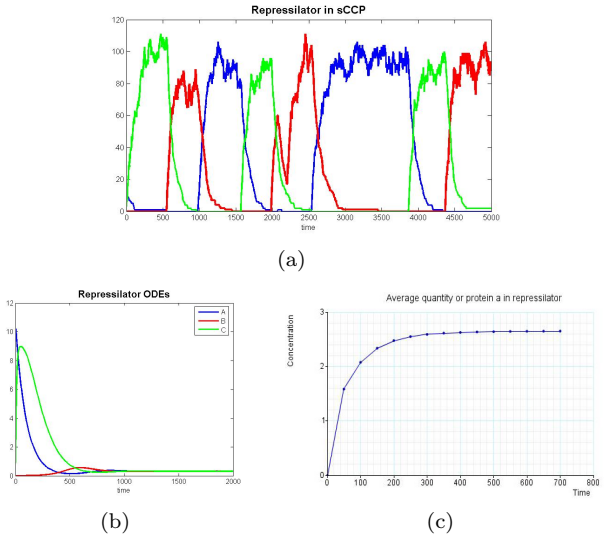


Fig. 1. **1(a)** Stochastic time trace for the Repressilator system of Table 2. Parameters are $k_p = 1$, $k_d = 0.01$, $k_b = 1$, $k_u = 0.01$. **1(b)** Solution of the differential equations of Table 3, automatically derived from sCCP program of Table 2. Parameters are the same as in stochastic simulation. **1(c)** Average trace of the stochastic system for one of the three repressors, computed using PRISM [Kwiatkowska et al., 2004].

We can now apply the translation procedure discussed in Section 2.2 to this particular model. First of all, we have to generate the reduced transition systems for all the agents of the network. They are shown in Figure 2. As we can see, the degradation agents have an RTS with just one state, while the NEG gates have two states, one corresponding to the active state of the gene, and the other one to the repressed state of the gene. The ODEs derived from these RTS are shown in Figure 3, while their numerical integration is shown in Figure 1(b). As we can readily see, there is no oscillation at all, but rather the three proteins converge to an asymptotic value, after an initial adjustment. In Blossey et al. [2007], the authors give arguments suggesting that the equations in Figure 3 never oscillate, whatever the value of parameters.

Inspecting the ODEs, we note the presence of six variables in addition to those representing the quantity of repressors in the systems. Such variables correspond to RTS-states of genes gates and they are used to model the configuration's change of the gates, from active to repressed and viceversa. The scenario looks rather suspicious: there is no argument to support the introduction of the additional variables, especially because we are continuously approximating boolean quantities (i.e. the on/off state of genes).

We stress the fact that the presence of these six additional variables is not a mere byproduct of the translation of Section 2.2, but rather they emerge naturally as a consequence of the structure of the model. A NEG gate, in fact, corresponds to the following set of reactions [Blossey et al., 2007]: $g \to g + P$, $g + R \to g'$, $g' \to g$, $P \to \emptyset$. If one writes the standard mass action equations for the reactions of the three NEG gates defining the Repressilator, one will obtain exactly the equations of Table 3.

As a matter of fact, the situation with this model of Repressilator is even more puzzling: also the average trace of the stochastic model shows no oscillations at all! This trace, obtained using transient analysis techniques implemented in PRISM [Kwiatkowska et al., 2004], is shown in Figure 1(c). What happens can be readily explained: the stochastic traces show an high variability in the duration of oscillations, so that the phases of two traces are uncorrelated at regime. Therefore, fixing a time instant, in two thirds of the traces each protein will not be expressed, while in the other third of the traces it will be at its peak value (equal to k_p/k_d). Hence, in the average, the quantity of a protein converges to $\frac{k_p}{3k_d}$.

Averaging the system or introducing continuous state variables, we destroy the information regarding the sequence of changes of gene's states. We conjecture that the oscillatory behavior of Repressilator is induced *exactly* by the switching dynamics of genes.

In order to preserve the information of RTS-states we translate the (d)sCCP program into an hybrid automaton, whose discrete modes correspond to *combinations* of RTS-states of the system's components. It will turn out that this move is enough to restore the oscillating behavior of Repressilator (thereby supporting our previous conjecture).

4. SCCP TO HYBRID AUTOMATA

The translation of a (d)sCCP network $\mathcal{N}$ to a hybrid automaton proceeds in two phases: first, each sequential component A_i of $\mathcal{N}$ is converted into a hybrid automaton,

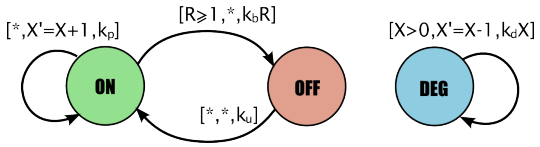


Fig. 2. Reduced transition systems for the NEG and the degrade sCCP agents of Table 2. X and R are template variables as in Table 2.

$$\begin{aligned}\dot{A} &= k_p Y_A - k_d A \\ \dot{B} &= k_p Y_B - k_d B \\ \dot{C} &= k_p Y_C - k_d C \\ \dot{Y}_A &= k_u Z_A - k_b Y_A C \\ \dot{Y}_B &= k_u Z_B - k_b Y_B A\end{aligned} \qquad \begin{aligned}\dot{Y}_C &= k_u Z_C - k_b Y_C B \\ \dot{Z}_A &= k_b Y_A C - k_u Z_A \\ \dot{Z}_B &= k_b Y_B A - k_u Z_B \\ \dot{Z}_C &= k_b Y_C B - k_u Z_C\end{aligned}$$

Fig. 3. ODEs derived for the Repressilator, generated by the method of Section 2.2. Variables Y_X and Z_X, $X = A, B, C$ are respectively RTS-state variables for active and inactive states of gene X.

then these HA are "glued" together using a suitable *product of automata* construction.

Let us outline the general technique. Consider a sequential agent A (cf. Def. 2) and construct its $RTS(A)$. The control graph of the hybrid automaton $HA(A)$ associated to A will correspond to the graph of the RTS, with loop transitions removed. The flows within each mode will be determined by suitably specializing the method of Section 2.2. Defining conditions on HA-edges, instead, is a more complicated matter, as it requires to look into the semantics of stochastic sCCP-transitions.

Consider the set of RTS-edges $E(A)$, and let $\sigma \in S(A)$ be an RTS-state. Then $E(A, \sigma)$ denotes the subset of edges looping in σ, and $E_l(A) = \bigcup_{\sigma \in S(A)} E(A, \sigma)$. In addition, the set of non-looping transitions is denoted by $E_0(A) = E(A) \setminus E_l(A)$.

As hinted above, the control graph of the hybrid automaton $HA(A)$ is $G = (S(A), E_0(A))$, i.e. its nodes are the RTS-states of $S(A)$ and its edges are the non-looping edges $E_0(A)$. The variables of $HA(A)$, instead, comprehend the variables $\mathbf{X}$ of the store, considered as continuous and evolving according to specific differential equations. These are constructed specializing the general method of Section 2.2: to every RTS-state $\sigma \in S(A)$, we associate a *local interaction matrix*, restricted to variable updating of RTS-edges $e \in E(A, \sigma)$ *looping in* σ. This matrix, call it I_σ, has one row for each variable $\mathbf{X}$, and one column for each $e \in E(A, \sigma)$. In every column, there is an entry k in correspondence to a variable X_j, if the update of e states that $X'_j = X_j + k$; all other entries are equal to zero. Moreover, the entries of the *rate vector* r_σ, of size $|E(A, \sigma)|$, are obtained by multiplying the rate function $rate(e)$ by the indicator function of $guard(e)$ (this vector differs from the rate vector of Section 2.2 for the absence of RTS-state variables). Hence, the equation vector ode_σ in RTS-state σ is $ode_\sigma = I_\sigma \cdot r_\sigma$.

We now turn to the crucial definition of *activation conditions* of HA-edges. Here we need to take into account the temporal evolution of the system: while in the (d)sCCP program the temporal duration of events is governed by stochastic laws, in the automaton we need to remove the stochastic ingredient without altering too much the timing structure of events.

First of all, observe that part of the discrete events of the (d)sCCP program has been subsumed by the differential equations in each mode. Therefore, we need to deal only with the events triggering an internal RTS-state change, i.e. with RTS-edges in $E_0(A)$. To do this, we need a few auxiliary variables, one for each RTS-edge of $E_0(A)$. The transition variable associated to edge $e \in E_0(A)$ will be identified with Y_e.

In order to correctly regulate the timing of state-changing events, we need to set appropriate activation conditions implicitly depending on time. Consider an RTS-edge $e \in E_0(A)$: in sCCP the timing of the associated transition is governed by its rate $rate(e) = \lambda$. In general, λ is a function $\lambda(\mathbf{X})$ of (some) system variables. Actually, as these variables vary continuously over time, governed by differential equations, we have that λ is time-varying as well, say $\lambda = \lambda(t)$. The transition in the (d)sCCP program, when isolated from the context, is a *non-homogeneous Poisson process* [Ross, 1996]. Thus, we can define the *cumulative rate function*

$$\Lambda(t) = \int_{t_0}^{t} \lambda(s) ds,$$

which is a monotone function of t. The theory of non-homogeneous Poisson processes then states that the number of firings at time t behaves like a Poisson variable with rate equal to $\Lambda(t)$, hence *the average number of firings of the transition at time* t *equals* $\Lambda(t)$. Therefore, we may activate the transition whenever $\Lambda(t) \geq E[\Lambda(t)] = 1$, corresponding to the happening of *at least one firing on average*. This condition is expressed in the HA as $Y_e \geq 1$, with the associated transition variable Y_e evolving according to

$$\dot{Y}_e = \frac{d\Lambda(t)}{dt} = \lambda(\mathbf{X}). \tag{1}$$

Clearly, variables Y_e's are additional HA-variables, while their differential equations will become part of the flow conditions of each mode.

In the activation conditions for $e \in E_0(A)$ we also need to take into account the (d)sCCP transition's guards. Hence,

$$jump(e) = guard(e) \wedge Y_e \geq 1.$$

The choice of $\geq$ instead of a simpler equality is introduced in order to avoid *starvation* for the transition [4].

All discrete HA-transitions are required to be *urgent*, i.e. fired as soon as $jump(e)$ is true.

The resets of the HA-edges $e \in E_0(A)$ consist of two components. The first one coincides with the update of the corresponding RTS-edge, while the second sets to zero all variables Y_e—this is a sort of counterpart of the memoryless property of Markovian processes.

If the rate function $\lambda(\mathbf{X})$ is constant, $\lambda(\mathbf{X}) \equiv k$, then Equation (1) reduces to $\dot{Y}_e = k$, which has solution $Y_e(t) = kt$ (provided $Y_e(0) = 0$). Hence, $Y_e \geq 1$ can be rewritten as $t \geq \frac{1}{k}$: $\frac{1}{k}$ is the expected time of the first firing of the corresponding (d)sCCP transition. Essentially, a HA-transition fires after a time equal to the expected time of the corresponding transition in the stochastic (d)sCCP component.

The above discussion can be synthesized in the following:

[4] $guard(e)$ can be false when $Y_e = 1$, but true infinitely often later on.

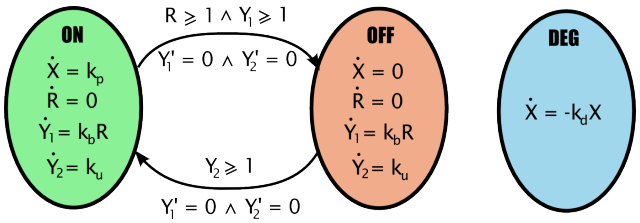


Fig. 4. Hybrid automata associated to Neg (left) and Degrade (right) agents of Table 2.

Definition 4. Let A be a (d)sCCP sequential agent operating on store variables $\mathbf{X}$. The hybrid automaton $HA(A) = (V, E, \mathbf{Z}, flow, init, inv, jump, reset)$ associated to A is defined by

(1) the control graph is given by $V = S(A)$ and $E = E_0(A)$;
(2) the variables are $\mathbf{Z} = \mathbf{X} \cup \{Y_e \mid e \in E\}$;
(3) for each $v \in V$, $flow(v)|_X = ode_v|_X$ if $X \in \mathbf{X}$, otherwise $X = Y_e$ for some $e \in E$ and $flow(v)|_X = rate(e)$ [5];
(4) $init(v)$ is $(\mathbf{X} = \mathbf{x_0}) \wedge \big(\bigwedge_{e \in E} Y_e = 0\big)$, for each vertex $V \in V$, where $\mathbf{x_0}$ is the initial value of $\mathbf{X}$ in the store (cf. Def. 1);
(5) $inv(v) = true$ for each $v \in V$;
(6) $jump(e) = guard(e) \wedge Y_e \geq 1$, for each $e \in E$;
(7) $reset(e) = update(e) \wedge \big(\bigwedge_{e \in E} Y_e' = 0\big)$, for each $e \in E$.

As an example, consider the (d)sCCP code of a Neg(X, R) gate (cf. Section 3), as defined in Table 2. Its RTS is shown in Figure 2 and consists of two states and three edges, one looping in RTS-state ON, the other two connecting back and forth RTS-states ON and OFF. The hybrid automaton associated to it is shown in Figure 4.
It has two modes and four variables, two corresponding to the store variables, X and R, and the other two, Y_1 and Y_2, associated to non-looping edges. Their differential equations are, according to Definition 4, $\dot{Y}_1 = k_i R$ and $\dot{Y}_2 = k_u$, for both modes. Equations for X and R, instead, depend on the mode: in the ON state they are $\dot{X} = k_p$ and $\dot{R} = 0$, while in the OFF state $\dot{X} = \dot{R} = 0$. In Figure 4 it is shown also the HA associated to the Degrade agent of Table 2: it has one mode, no edges, and one variable X evolving according to $\dot{X} = -k_d X$.

Automata (Flux) Product. Def. 4 gives a recipe to construct hybrid automata of sequential components. These have to be combined together to form the hybrid automaton of the network. The key problem is that the same variable of the store can be modified by several agents concurrently. To capture this feature at the HA level, we need a product construction which is able of superimposing fluxes, i.e. adding the right-hand side of the differential equations of each component for all shared variables. 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 5. (Flux Product). 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) $flow((v_1, v_2))|_X = flow_1(v_1)|_X + flow_2(v_2)|_X$, if $X \in \mathbf{X_1} \cap \mathbf{X_2}$. Otherwise, if $X \in \mathbf{X_i}$, then $flow((v_1, v_2))|_X = flow_i(v_i)|_X$;
(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 $\pi_2(e) = e_2$.

The flux product is almost the classical product of two HA [Henzinger, 1996], the only difference being the treatment of fluxes for variables shared among the factors. In our case, in fact, fluxes are added.
Inspecting the previous definition, it is not difficult to see the following:

Lemma 2. The flux product $\otimes$ is associative and commutative.

We are now ready to define the hybrid automaton associated to a (d)sCCP network $\mathcal{N}$:

Definition 6. Let $\mathcal{N}$ be a (d)sCCP network with components $A_1, \ldots A_n$. The hybrid automaton associated to $\mathcal{N}$ is

$$HA(\mathcal{N}) = HA(A_1) \otimes \ldots \otimes HA(A_n).$$

Notice that the modes of the automaton $HA(\mathcal{N})$ are the combination of modes of the sequential components; in total, $HA(\mathcal{N})$ has $|S(A_1)| \times \ldots \times |S(A_n)|$ states. In addition, the variables of $HA(\mathcal{N})$ are the store variables $\mathbf{X}$ of the system, shared by all components, for which fluxes are added, and the variables involved in the activation conditions of transitions, which are local within each component.

Repressilator revisited. It is time to go back to our initial example, i.e. the Repressilator. In Section 3 we showed that the simple stochastic model based on gene gates oscillates, while the corresponding ODE model, generated with the automatic translation method of Section 2.2, does not oscillate at all. From the analysis of the derived equations, we had the idea of preserving part of the discreteness of sCCP, translating it to hybrid models. The hope was that the preservation of the discrete behavior of gene gates was sufficient to produced the desired oscillatory behavior.

The sCCP model of Repressilator has six components, three gene gates and three agents degrading the three proteins. The HA for these components are shown in Figure 4; applying the flux product construction to them, we obtain a hybrid automaton with 8 modes, corresponding to the possible combinations of genes being on or off,

[5] $flow(v)|_X$ denotes the restriction of vector $flow(v)$ to the component corresponding to variable X.

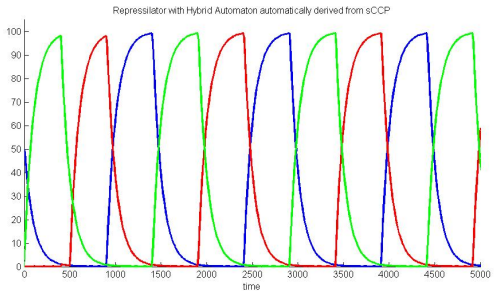


Fig. 5. Time trace of the hybrid automaton associated to the Repressilator system applying the method of Section 4. Parameters are the same as those in caption of Figure 1. State changing transition are activated here as soon as the integral functions of local time reach their threshold value. This causes the regularity of the oscillation, as opposed to the stochastic trace of Figure 1(a).

and 9 variables. Three of them, A, B, C, are those giving the quantity of proteins in the system, while the other six $Y_{X,on}, Y_{X,off}$, $X = A, B, C$, govern activation and deactivation of genes.
The equations in each mode are decoupled; for instance, when gene A is on, the equation for $\dot{A}$ is

$$\dot{A} = k_p - k_d A,$$

simplifying to

$$\dot{A} = -k_d A,$$

when the gene if off. Notice that these equations are obtained by adding the right-hand side of the equations for the Neg and Degrade HA (cf. Figure 4). The interactions between repressors and genes, instead, are confined to the activation conditions of the automaton transitions. Consider again gene A and suppose to be in a mode of the automaton where gene is on. Then, the transition switching this gene off has an activation condition equal to

$$Y_{A,off} \geq 1 \wedge C \geq 1,$$

with $\dot{Y}_{A,off} = k_b C$ depending from the repressor C ($C \geq 1$, instead, is the guard of the corresponding RTS-edge). The transition that turns gene A on, instead, has a constant rate k_u, hence its activation condition is $Y_{A,on} \geq 1$, $\dot{Y}_{A,on} = k_u$.
In Figure 5 we show a simulation of the resulting hybrid automaton, with urgent transitions. We can observe that the mixed discrete/continuous nature of the automaton is able to capture the oscillatory behavior.

As a final remark, note that in sCCP the guards $R \geq 1$ and $R > 0$ (where R is one of the three repressors A, B, C) are equivalent, as molecules have integer quantities. This is not true anymore in the hybrid automaton, as small values of the repressor still contribute to equation (1), and may activate the repression even if $R < 1$, a non-sense.

State-space explosion. One issue concerning the construction outlined in the paper is the number of discrete modes of the final HA. In fact, the flux product construction generates a number of modes exponential in the number of components. However, as long as we are interested in simulating the HA, this is not a big issue. In fact, not all putative modes are always necessary. In particular, the automaton can be *simulated on-line*, as in each temporal instant only the current mode and its outgoing transitions need to be kept in memory; the other modes can be inferred directly from the HA of each single component.

Non-determinism. In the simplest case, we require all transitions of the hybrid automaton to be *urgent*, meaning that they are taken whenever their guards are satisfied. In this way, transition are deterministic: they are executed in one single point of the time axis. Actually, although this point corresponds to the mean of the stochastic process, we are losing all the effects of stochastic variability. In order to recover part of this effect, we can introduce some *non-determinism*. The idea is simple: we can split an HA-edge e in two, both having the same activations and resets derived from guards and updates of the corresponding (d)sCCP transition, but differing in the timing conditions. Choosing an interval $[1-\delta_1, 1+\delta_2]$, $\delta_1 < 1$, centered around 1, we activate one transition non-deterministically when $Y_e \in [1-\delta_1, 1+\delta_2]$ (and the other guards are true). The other transition, instead, is urgent and it is active when $Y_e \geq 1+\delta_2$. This second transitions is introduced in order to deal with situations when the first transition can never be taken due to the presence of false guards when $Y_e \in [1-\delta_1, 1+\delta_2]$. The size $\delta_1 + \delta_2$ of $[1-\delta_1, 1+\delta_2]$ essentially controls how much of the stochastic variability we are capturing; in fact, the fraction of stochastic transitions we are capturing equals the probability of transition time T belonging to interval $[\frac{1}{\lambda} - \delta_1, \frac{1}{\lambda} + \delta_2]$. Reasonably, the size $\delta_1 + \delta_2$ cannot be too large, as all times in $[\frac{1}{\lambda} - \delta_1, \frac{1}{\lambda} + \delta_2]$ are to be considered equally probable to be chosen in the non-deterministic case (in contrast with the stochastic model, where they are exponentially distributed).

5. CONCLUSIONS

We focused on the problem of approximating stochastic dynamics of process algebras with continuous dynamical systems. Instead of working with differential equations, we used hybrid systems, mixing discrete and continuous evolution. The discrete ingredient that we try to preserve in this translation is related to all the structures devoted to control the flow of events in process algebras. Essentially, we want to represent explicitly those behaviors depending on an inner state of a process. The example we used is that of a negative gene gate, which can be in two different states: *on*, producing the coded protein, or *off*, not producing anything. In this translation, the difficult part consists in defining coherently the activation regions of automaton transitions: we used conditions based the theory of non-homogeneous Poisson processes. We then applied the method to Repressilator, a system composed by three negative gene gates, cyclically repressing each other, showing that the resulting automaton shows stable oscillatory behavior, in contrast with the ODEs obtained by methods present in literature.

The main problem constantly in the background of this work is that of finding automatic translation procedures from SPA to ODE's or hybrid systems, *preserving the observed behavior* of the described systems. The methods

present up to know in literature map SPA to ODE's using purely syntactic criteria, and in several situations fail to preserve the dynamics. There are several sources of non-equivalence, which can be broadly divided into three classes: the effect of stochastic fluctuations, the bad approximations done in passing from discrete to continuous variables (especially when the discrete variables are small) and the description of logical structures of control with continuous variables. We introduced hybrid automata to deal precisely with this last class of disturbance, describing these control structures explicitly in the control graph of the target hybrid automaton. Therefore, this approach seems suited to deal especially with all those systems that can be described as *circuits*, whose components are switches or other logical devices, like *genetic networks*. This intuition is supported by several tests we performed on other genetic networks described by gene gates (results not shown).
Unfortunately, the other two sources of non-equivalence are still there, though the framework of hybrid systems can provide interesting tools to cope with them. One possibility to deal (qualitatively) with stochastic effects is that of using non-determinism to let hybrid automata produce set of traces showing a certain degree of variability. For instance, non-determinism for the Repressilator may result in oscillations with variable period, pretty much what happens in its stochastic version. The bad approximation of low populations, instead, may be dealt by retaking and extending the observation made in Alur et al. [2001]: the dynamics of our systems should be continuous for big populations and discrete for small ones. Actually, in Alur et al. [2001] authors suggest to use a stochastic dynamics at low regimes, though the treatment of the stochastic ingredient we propose here results in systems that are not stochastic, hence computationally easier to analyze. More concretely, for every variable of the system we can define two modes: one in which dynamics is continuous and the other where the evolution is discrete, i.e. represented by a looping HA-transition with activation conditions defined similarly to those of Section 4.

We would like to make a couple of final remarks. First of all, the treatment of the stochasticity by means of non-determinism is somewhat unusual: normally, in process algebras non-determinism is quantitatively "solved" using some stochastic ingredient, while here we use non-determinism to qualitatively encode stochastic effects. Secondly, comparing this translation with the methods associating ODEs to SPA [Bortolussi and Policriti, 2007, Hillston, 2005, Cardelli, 2006], we observe that while these methods are based on a purely syntactic manipulation of terms, our translation incorporates non-trivial semantical ingredients in the way transitions of the automaton are defined.

We conclude by noting that throughout the paper we used the vague locution "behavioral equivalence", without defining it precisely. This is by itself an interesting problem, as it corresponds to decide what precisely we want to capture and what we are willing to abandon in a translation procedure. We believe that, above all, the basic ingredient we need to preserve is the qualitative evolution of the system, having maybe a family of stricter equivalences gradually taking into account quantitative ingredients. In this direction, we are currently investigating approaches based on the use of Temporal Logic.

REFERENCES

R. Alur, C. Belta, F. Ivancic, V. Kumar, M. Mintz, G. Pappas, H. Rubin, and J. Schug. Hybrid modeling and simulation of biomolecular networks. In *Proceedings of HSCC*, LNCS 2034, pages 19–32, 2001.

M. Antoniotti, A. Policriti, N. Ugel, and B. Mishra. Model building and model checking for biochemical processes. *Cell Biochemistry and Biophysics*, 38(3):271–286, 2003.

R. Blossey, L. Cardelli, and A. Phillips. A compositional approach to the stochastic dynamics of gene networks. *T. Comp. Sys. Biology*, pages 99–122, 2006.

R. Blossey, L. Cardelli, and A. Phillips. Compositionality, stochasticity and cooperativity in dynamic models of gene regulation. *HFPS Journal, in print*, 2007.

L. Bortolussi. Stochastic concurrent constraint programming. In *Proceedings of QAPL 2006, ENTCS*, volume 164, pages 65–80, 2006.

L. Bortolussi. *Constraint-based approaches to stochastic dynamics of biological systems.* PhD thesis in Computer Science, Univ. of Udine, 2007. `http://www.dmi.units.it/~bortolu/files/reps/Bortolussi-PhDThesis.pdf`.

L. Bortolussi and A. Policriti. Modeling biological systems in concurrent constraint programming. *Constraints*,13(1), 2008.

L. Bortolussi and A. Policriti. Stochastic concurrent constraint programming and differential equations. In *Proceedings of QAPL 2007*, 2007.

L. Cardelli. From processes to odes by chemistry. *downloadable from* `http://lucacardelli.name/`, 2006.

M.B. Elowitz and S. Leibler. A synthetic oscillatory network of transcriptional regulators. *Nature*, 403:335–338, 2000.

D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. of Physical Chemistry*, 81(25), 1977.

T. A. Henzinger. The theory of hybrid automata. In *Proceedings of LICS '96*, 1996.

J. Hillston. Fluid flow approximation of pepa models. In *Proceedings of QEST 2005*, 2005.

H. Kitano. *Foundations of Systems Biology.* MIT Press, 2001.

H. Kitano. Computational systems biology. *Nature*, 420: 206–210, 2002.

M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with prism: A hybrid approach. *Int. Jo. on Software Tools for Technology Transfer*, 6(2):128–142, September 2004.

J. R. Norris. *Markov Chains.* Cambridge University Press, 1997.

C. Priami and P. Quaglia. Modelling the dynamics of biosystems. *Briefings in Bioinformatics*, 5(3):259–269, 2004.

A. Regev and E. Shapiro. Cellular abstractions: Cells as computation. *Nature*, 419, 2002.

S. M. Ross. *Stochastic Processes.* Wiley, New York, 1996.

V. A. Saraswat. *Concurrent Constraint Programming.* MIT press, 1993.

D. J. Wilkinson. *Stochastic Modelling for Systems Biology.* Chapman & Hall, 2006.