Perspectives on Constraints, Process Algebras, and Hybrid Systems

Luca Bortolussi
Dept. of Mathematics and Informatics, University of Trieste, Italy.
luca@dmi.units.it
and Alberto Policriti
Dept. of Mathematics and Informatics, University of Udine, Italy.
Istituto di Genomica Applicata, Udine, Italy.
policriti@dimi.uniud.it

Abstract

Building on a technique for associating Hybrid Systems (HS) to stochastic programs written in a stochastic extension of Concurrent Constraint Programming (sCCP), we will discuss several aspects of performing such association. In particular, as we proved an sCCP program can be mapped in a HS varying in a lattice at a level depending on the amount of actions to be simulated continuously, we will discuss what are the problems involved in a semi-automatic choice of such level. Decidability, semantic, and efficiency issues will be taken into account, with special emphasis on their links with biological applications. We will also discuss about the role of constraints and of the constraint store in this construction.

1 Introduction

Systems biology emerged in recent years as the discipline promising to deepen the understanding of living beings, studying them from a systemic perspective. Interactions among constituents are considered in their concerted activity, and biological behavior is seen as an emergent property of these intricate patterns of cooperation and repression [13].

However, studying biological systems from this perspective is rather difficult for many reasons: the number of actions and interactions into play is huge, most of them are still unknown or poorly understood, the complexity of mathematical descriptions grows combinatorially, and efficiency and precision issues of models are critical.

Stochastic process algebras (SPA) [17], despite coming from the different context of performance analysis of distributed computing systems, proved to be a promising tool, inasmuch they offer a simple, compositional, description language that is automatically mapped into the complex mathematical formalism of continuous time stochastic processes, for simulation and analysis. Indeed, SPA have at their disposal automatic reasoning tools both at the syntactic and semantic levels, making them a powerful framework.

Commonly used SPA in systems biology are stochastic π-calculus [8] and bioPEPA [9]. However, their simple primitives may make difficult the description of complex interactions, like those typical of combinatorial biochemical networks [14]. Furthermore, they lack any ready-to-use computational or reasoning power, which is a limitation when facing the issue of modeling systems at greater level of detail.

Concurrent Constraint Programming [18], in its stochastic variant, **sCCP** [3], can be a way to circumvent these problems. In fact, it combines the simplicity of process algebras for describing agents with the reasoning and computing capabilities of constraints.

In our attempt to use **sCCP** as a modeling framework for biological systems, we found two main advantages:

1. the "computational twist" of constraints allow a compact description of complex operations, as those needed to describe combinatorial biochemical networks [2] or spatiality [6], without the need of introducing further primitives in the language.

2. The separation between agents and the constraint store (cf. Section 2) forces a modeling discipline that requires to separate the description of the *control logic* of the system (modeled by agents) from the description of the *configurations* of the system (naturally modeled in the constraint store).

This second aspect proved its utility as it greatly simplified the definition of an additional semantics for **sCCP** in terms of (a family of) Hybrid Automata [5], thus enhancing the mathematical instruments at disposal for analysis

However, these two features of **sCCP**, namely the "computational twist" and the flexible semantics based on hybrid automata, are somehow in conflict. In fact, the first one exploits the reasoning power of the constraint store, while the second one works when constraints are simplified to very basic interactions. Reconciling these aspects is an open problem.

In addition to the above issues, our reflection on the perspectives of usage of **sCCP** for Systems Biology must take into account realistic and effective semantics for **sCCP**. If, on the one hand, the control mechanisms are naturally interpreted in fully discrete terms, on the other hand, the stochastic as well as some of the substance-level modelling (constraint) variables, are more naturally kept continuous. This was the main reason that lead us to the above mentioned Hybrid Automata semantics, which can be seen as a way to associate **sCCP** programs to Hybrid Automata organized in a *lattice* at varying levels of discreteness.

Some of the features of such (variable) association were expected: the higher the level of discreteness to be maintained, the more adherent the program behaviour to the automaton time-evolution to be observed. Some other feature we found surprising: stochasticity can sometimes be dropped in favor of discreteness alone, with a very positive drawback in terms of simulation costs.

We conclude our discussion here putting forward a pair of open problems naturally arising in the above outlined framework:

1. Is there an *optimal* level of discreteness to be maintained when associating a hybrid automaton with a(n **sCCP**) program?

2. Can we automatically or semi-automatically address the above question by some sort of analysis of the (**sCCP**) program?

In this paper, we will not provide answers to such questions, but rather we will discuss these open problems in more detail, suggesting possible directions of attack. Before doing this, we will briefly survey the previous work on **sCCP** and on the hybrid semantics in a non-formal manner, illustrating the relevant notions by means of an example.

2 Stochastic Concurrent Constraint Programming

sCCP [3] has two basic ingredients: *agents* and *constraints*. Agents are the main actors, interacting by asynchronously exchanging information in form of *constraints*, through the *constraint store*. **sCCP** has been mainly applied as a modeling language for biological systems [3], using the constraint store to describe the state of the system, e.g. numerosity of molecular species. As these quantities evolve in time (and one is precisely interested in such a dynamics), we considered special variables, called *stream variables*, which can change value during computation (in contrast with standard logical variables, that can be instantiated just once). At least for modelling simple biological scenarios, one needs very simple constraints, basically comparing and assigning new values to stream variables.

As an example, consider a simple genetic network, with one single gene producing a protein X at a basal rate k_p, acting as its own repressor (by binding in the promoter region of the gene). When the gene is repressed, it does not produce protein X and, after a delay, it goes back in the normal state (i.e., the repressor unbinds from the gene). Each copy of protein X is also constantly subject to degradation at rate k_d.

In order to model such a system in **sCCP**, we need one stream variable keeping track of the amount of protein X in the system. All interactions, instead, are described by **sCCP** agents. In particular, we need a simple recursively looping agent to model degradation and an agent modelling the gene. This latter agent is slightly more complex, as it describes the control mechanisms that the gene is subject to.

The **sCCP** program is `gene_on` $\parallel$ `degrade`, where (* stands for true):

$$\begin{array}{lcl} \texttt{gene\_on} & \stackrel{\text{def}}{=} & [* \to X' = X + 1]_{k_p} \texttt{.gene\_on} + [X > 0 \to *]_{k_b X} \texttt{.gene\_off} \\ \texttt{gene\_off} & \stackrel{\text{def}}{=} & [* \to *]_{k_u} \texttt{.gene\_on} \\ \texttt{degrade} & \stackrel{\text{def}}{=} & [X > 0 \to X' = X - 1]_{k_d X} \texttt{.degrade} \end{array}$$

The basic actions executable by the agents above are *guarded updates* of the form $[G \to R]_\lambda$, where G is a *guard* that must be satisfied for the action to be performed and R is the *update* policy—basically a conjunction of atoms of the form $X' = X + k$. Furthermore, each action has a *stochastic duration*, given by an exponentially distributed random variable with rate depending on the state of the system through a positive real-valued function λ. Additionally, the language has standard constructs of SPA: stochastic choice $+$, parallel composition $\parallel$, and recursion.
The semantics of **sCCP** is given by a Continuous Time Markov Chain [16] (CTMC). Definitions and further details can be found in [3].

Remark 2.1. The constraints that can be used to update the constraint store are rather limited, as they simply add a constant term to some stream variables. This restriction, however, allows to interpret **sCCP**-actions as continuous fluxes, a required condition to define the hybrid semantics, see also Section 4. To model more involved biological systems, more complex update constraints and more complex constraint stores can/should be considered. For instance, in [2], we used a class of constraints operating on graphs and stream variables to tame the combinatorial complexity of modelling the formation of protein complexes. We will return on the issue of interfacing these two needs in Section 5.

Looking again at the example, we can see that the parallel operator is used only to compose the single agents, but not within agents. When such condition is in force, we can represent **sCCP** agents as automata, synchronizing on store variables, called *Reduced Transition Systems* (RTS) [4]. The RTS of the agents of the example are shown in Figure 1(a). As can be seen, recursion is basically dealt with by introducing loops in the graphs, whose edges are labelled by rate/guard/update of the corresponding **sCCP** action. In Figure 1(a), there are also two additional variables: G_1 and G_0. They keep track of which state of the agent is the active one, with $G_1 = 1$ and $G_0 = 0$ corresponding to `gene_on` and $G_1 = 0$ and $G_0 = 1$ to `gene_off`. The updates of edges deal with such variables mimicking the program structure, while explicit dependence on G_0 and G_1 is introduced in rates (i.e. production rate becomes $k_p G_1$, because production is possible only in state `gene_on`). This is a technical trick useful to introduce the hybrid semantics.

3 Hybrid Automata

The hybrid semantics of **sCCP** will be defined in terms of Hybrid Automata (HA), see [11] for more details.

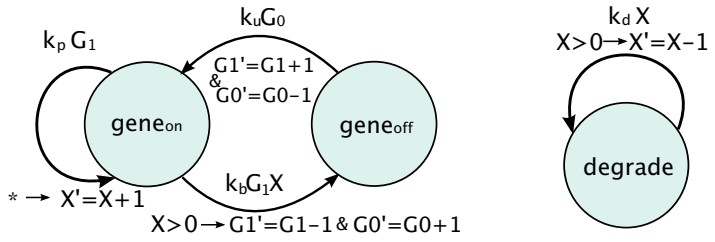
The basic idea of HA is that they have a mixed discrete/continuous evolution. The discrete part of the system is described as a labelled graph, while the continuous part is modelled by an array of real-valued variables $\mathbf{X}$. In each vertex q of such a graph, called *mode*, variables are subject to a continuous evolution, usually defined by a set of ordinary differential equations (ODE) $\dot{\mathbf{X}} = F_q(\mathbf{X})$. The continuous evolution within a mode can be interrupted by the happening of a discrete event, corresponding to an edge of the graph. This event happens as soon as specific conditions on variables (described by a guard predicate) becomes true. Its execution changes the mode of the automaton (hence, also the ODE may change) and modifies discontinuously the value of variables $\mathbf{X}$, according to an edge-dependent reset policy.
Usually, compositionality of HA is achieved by a suitable definition of a HA-product, cf. [11, 7].

4 Hybrid Semantics of sCCP

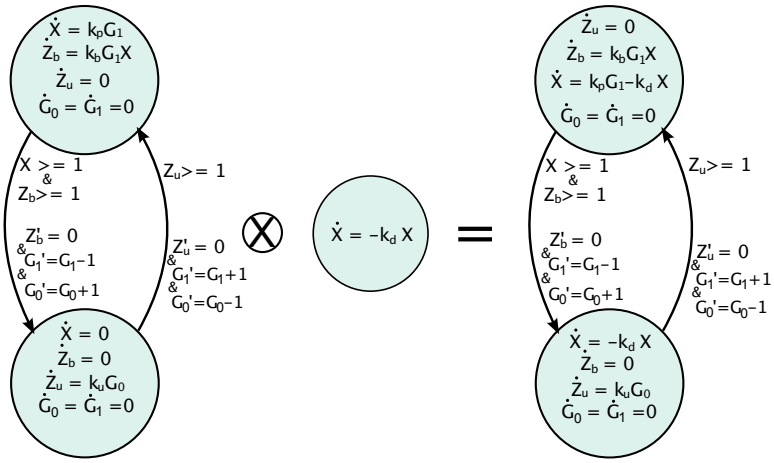
In this section we informally explain the definition of the hybrid semantics for **sCCP** [5, 7]. The construction is compositional: first, single **sCCP** agents are converted into HA, then these HA are combined by taking their product.

The mapping starts from the RTS of each **sCCP** agent. The first step consists in partitioning the edges of such a RTS into two sets: those that will contribute to continuous dynamics and those that will define the discrete skeleton.
Consider again the RTS of the gene agent of Figure 1(a). It has three edges: we will treat as continuous only the looping edge on `gene_on`, corresponding to the production of X, while the ones corresponding to binding and unbinding will be dealt discretely. This is nothing but one possible choice. Think, for instance, of the case in which *all* edges are treated as continuous. Actually, all possible partitions are admissible, and the final choice is left to the modeler.
Once the edges are partitioned, we can construct the graph of the hybrid automaton. This is essentially derived from the RTS, collapsing nodes connected by continuous transitions and removing edges to be treated as continuous.
The continuous dynamics within each mode is defined according to continuous transitions connecting collapsed RTS-states. Consider the continuous transition producing protein X. It modifies only variable X, increasing it by 1 unit, with rate $k_P G_1$. The associated ODE is $\dot{X} = (+1) \cdot k_P G_1$, an it can be seen as obtained by multiplying the net increase of X by the rate. In case more than one transition is acting on a variable, their effect will be summed up.[1]
The definition of the discrete dynamics, instead, is slightly more complicated. In fact, we need to render the fact that **sCCP** actions *take time to be executed*. This is somehow un-natural for HA, in which discrete transitions are instantaneous. The idea is to introduce extra (continuous) variables to faithfully govern firing of discrete transitions. Such firing will happen when a threshold value set at the expected time of the stochastic transition is reached. Consider the RTS-edge corresponding to the repression of the gene in our running example. As discussed in [7], we can introduce a new continuous variable—Z_b in this case—and let it evolve according to $\dot{Z_b} = k_b G_1 X$, i.e. according to the rate of the stochastic transition. When Z_b reaches 1, we fire the transition and reset Z_b to zero. Z_b can be seen as a clock evolving at a non-constant speed.

[1] This is the motivation for requiring constant updates of variables. In fact, it is not clear how to describe in terms of continuous fluxes other kind of updates, think for instance at $X' = k$.



(a) Reduced Transition System



(b) Hybrid Automaton

Figure 1: (bf1(a)) Reduced Transition System for the agents of Section 2. (bf1(b)) Hybrid Automata obtained from gene example. Variables Z_b and Z_u are associated with the edge from gene_{on} to gene_{off} and from gene_{off} to gene_{on} of the RTS. See the text for a more detailed discussion on these edge variables.

In addition, each edge in the HA will be subject to the same guard and update policies of the corresponding **sCCP** action.
Once an HA has been build for each **sCCP** agent, these are composed together by a special product construction, which adds the right end sided of differential equations, cf. [7] for further details. The HA obtained for our example is shown in Figure 1(b).

The Lattice of HA. The previous construction is parametric with respect to the partitioning of **sCCP** actions into discrete and continuous. We can arrange the different HA so obtained in a lattice, where at the top element all **sCCP** actions are treated as continuous, while at the bottom element they are all kept discrete. Essentially, the fully continuous HA corresponds to the set of ODE associated to an **sCCP** program by fluid-flow approximation [4], while the fully discrete HA is a timed automata with skewed clocks (the so called Multi-Rate Timed Automata, [12]).

In this section, we assumed that the HA obtained has a (non-)deterministic evolution. We can also maintain the discrete dynamics stochastic, by simply replacing the threshold 1 in the guards of variables associated to edges by a randomly chosen threshold (exponentially

distributed with rate 1) [7].

5 Perspectives

In the introduction, we argued that **sCCP** has two characterizing features that are somehow in conflict. On the one side, in order to model more complex systems, we would like to increase the complexity of constraint-based operations acting on the constraint store. On the other side, however, we want a simple form for such interactions in order to use hybrid-automata based semantics.

How can we reconcile these two aspects? Considering the use of constraints of [2, 6], we can observe that, in all cases, the basic entity involved in modeling are stream variables. Constraints build a structure upon them to define complex manipulations and bookkeeping, in order to execute a sequence of simple operations as a single step activity.

One possibility is, therefore, to "make explicit" the constraints used, finding a low level description of the constraint store based only on stream variables, and precisely define the effect of each constraint in this new store. This construction can be hampered by combinatorial explosion of store size and even by the emergence of infinity in order to deal correctly with recursion. However, this direction is worth investigating, for it would reconcile these two conflicting aspects of **sCCP**.

Moving forward to the two open problems stated at the end of our introduction, let us observe that at the top of the lattice mentioned at the end of the previous section we have a single-mode automaton whose evolution is entirely described by a set of ODEs. Such an automaton can be seen as a full "mathematical" reduction of our initial **sCCP** program: being able to solve the ODEs we would have a complete solution of the system simulated by the **sCCP** program. The discrete dimension plays no role at the top of the lattice.

An attempt to push as down as possible this property, motivates us in the following definition:

Definition 5.1. An hybrid automaton $\mathcal{H}$ in the lattice associated to an **sCCP** program A is said to be an *optimal approximation* of A if and only if its bisimulation quotient[2] [15] is finite and every $\mathcal{H}'$ below $\mathcal{H}$ in the lattice does not enjoy this property.

On the ground of the previous discussion we have that given an **sCCP** program A there always exists a (not necessarily unique) optimal approximation of A.

Notice that it is not clear the role of stochasticity in Definition (5.1). In fact, we need to explain in some more detail the relationships intervening among continuity, discreteness and stochasticity in our construction. The key point is that, CTMC *are* decidable, in the sense that reachability is computable (one can compute the probability of reaching any subset of states with arbitrary precision), and model checking of CTL [10] formulae (or better, its stochastic version CSL [1]) is decidable. However, when we simplify a CTMC, replacing it by a Multi-Rate Automaton (MRA)[3], decidability is lost. This phenomenon is a consequence of the fact that in CTMCs time enters the picture orthogonally with respect to evolution: The choice of next state

[2]The bisimulation quotient of an hybrid automaton $\mathcal{H}$ can be seen statically or dynamically. Statically, is the coarsest partition refinement of the infinite state system (whose states are points in the n-dimensional space of flows), stable with respect to $\mathcal{H}$'s continuous and discrete transitions. Dynamically, is the fix point of the partitioning procedure splitting states with respect to the predecessor relation of arcs in $\mathcal{H}$.

[3]MRA correspond to CTMC in the lattive of non-stochastic HA, as they are the HA associated to the fully discrete case.

and the elapsed time before reaching it are probabilistically independent. In MRA, instead, time drives the evolution and the infinite precision involved in its density may result in the high expressiveness leading to undecidability.

The above discussion suggests that we can can tackle the open problems we proposed also focusing on the interplay among continuity, discreteness, and stochasticity. Alternatively, we can restrict our analysis on the removal of stochasticity, perhaps studying the effect of trading non-determinism and probability in our models. Again, a precise assessment of the level of decidability becomes an important benchmark for the approach.

As a final consideration we briefly comment on the following issue: is this circle of ideas/problems peculiar of Systems Biology? Biological systems seem naturally described by Hybrid Automata at a certain level of abstraction. At the finest level they can be modeled by CTMC, but these models may be not manageable in practice because of their complexity. However, the efficiency issue is not the most peculiar one for Systems Biology applications. In a certain sense, the features of biological systems hinting more directly at the necessity of a study of the above mentioned interplay, are their inherent uncertainty (consequence of realistic quantitative environmental interaction) and their complexity (consequence of our lack of knowledge of the internal biological control mechanisms). In this perspective, Systems Biology can be seen as a most interesting and promising arena in which testing model building techniques to mix different levels of discrete, continuous, and stochastic components.

References

[1] A. Aziz, V. Singhal, F. Balarin, R. Brayton, and A. Sangiovanni-Vincentelli. Verifying continuous time markov chains. In *Proceedings of CAV96*, 1996.

[2] L. Bortolussi, S. Fonda, and A. Policriti. Constraint-based simulation of biological systems described by molecular interaction maps. In *Proceedings of IEEE conference on Bioinformatics and Biomedicine, BIBM 2007*, 2007.

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

[4] L. Bortolussi and A. Policriti. Dynamical systems and stochastic programming — from ordinary differential equations and back. *Transactions of Computational Systems Biology*, 2009.

[5] L. Bortolussi and A. Policriti. Stochastic programs and hybrid automata for (biological) modeling. In *Proceedings of CiE 2009*, 2009.

[6] L. Bortolussi and A. Policriti. Tales of spatiality in stochastic concurrent constraint programming. In *Proceedings of BioLogic09*, 2009.

[7] L. Bortolussi and A. Policriti. Hybrid dynamics of stochastic programs. *Theoretical Computer Science*, 2010.

[8] L. Cardelli. Abstract machines of systems biology. *Transactions on Computational Systems Biology*, III, LNBI 3737:145–168, 2005.

[9] F. Ciocchetta and J. Hillston. Bio-PEPA: an extension of the process algebra PEPA for biochemical networks. In *Proceeding of FBTC 2007*, 2007. Workshop of CONCUR 2007.

[10] E. Clarke, A. Peled, and A. Grunberg. *Model Checking*. MIT press, 1999.

[11] T. A. Henzinger. The theory of hybrid automata. In *LICS '96: Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, 1996.

[12] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *Proceedings of 27th annual ACM symposium on Theory of Computing*, 1995.

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

[14] K. W. Kohn, M. I. Aladjem, J. N. Weinstein, and Y. Pommier. Molecular interaction maps of bioregulatory networks: A general rubric for systems biology. *Molecular Biology of the Cell*, 17(1):1–13, 2006.

[15] G. Lafferriere, G.J. Pappas, and S. Sastry. O-minimal hybrid systems. In *Proceedings of Mathematics of Control, Signals, and Systems (MCSS)*, 2000.

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

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

[18] V. Saraswat and M. Rinard. Concurrent constraint programming. In *Proceedings of 18th Symposium on Principles Of Programming Languages (POPL)*, 1990.