Languages for Concurrency Featuring Quantitative Information: An Overview and New Perspectives [⋆]

Jesús Aranda[1,2] and Jorge A. Pérez[3]

[1] INRIA Futurs and LIX, École Polytechnique, France
jesus.aranda@lix.polytechnique.fr
[2] Escuela de Ingeniería de Sistemas y Computación, Universidad del Valle, Colombia
[3] Dept. of Computer Science, University of Bologna, Italy
perez@cs.unibo.it

Abstract. The study of quantitative information within languages for concurrency has recently gained a lot of momentum. In many applications, quantitative information becomes crucial when refining models with empirical data, and is of the essence for verification purposes. In this paper we survey some of the existing languages for concurrency that feature quantitative information, with a special interest in those proposed for biological applications. This survey is then used as a context to motivate a novel approach for analyzing systems exhibiting stochastic behavior, in the form of a discrete-timed concurrent constraint process calculus. Some design decisions involved in the definition of an operational semantics for such a calculus are discussed.

1 Introduction

The study of *quantitative information* within languages for concurrency has recently gained a lot of momentum. In many applications, quantitative information becomes crucial when refining models with empirical data, and is of the essence for verification purposes. Two main models of quantitative information can be singled out from the vast literature on the subject. Given a computation that can perform different, competing actions, a *probabilistic* model provides a probability distribution over such actions. In contrast, a *stochastic* model relates each action to a random variable which determines its *duration*: given a set of competing actions, the fastest action (i.e. the one with the shortest duration) is executed. Consequently, notions not considered in a probabilistic model (e.g. speed) are fundamental in a stochastic setting. Not surprisingly, areas in which time is essential (e.g. systems biology, performance modeling) have found in languages featuring stochastic information adequate frameworks for analysis.

In the first part of this paper we review some languages for concurrency that include quantitative behavior in their syntax and/or semantics. Our review shall focus on

[⋆] Research partially supported by the COLCIENCIAS project REACT (No. 1251-330-18902) and the INRIA DREI Équipe Associée FORCES. The work of Jesús Aranda has been supported by COLCIENCIAS (Instituto Colombiano para el Desarrollo de la Ciencia y la Tecnología "Francisco José de Caldas"), INRIA Futurs and ÉGIDE (Centre français pour l'accueil et les échanges internationaux).

languages and formalisms proposed in the realm of *systems biology*. Within systems biology, languages for concurrency serve to develop a high-level representation that stands between the actual biological system and the mathematical models traditionally used for prediction purposes. Such a representation is intended as a testbed for plausible hypotheses; these are used both to complement real "in vivo" experiments and to refine the mathematical models. In turn, this could trigger better defined real experiments. Consequently, formal languages for systems biology not only must allow for faithful and intuitive way representations of systems, but should also offer mechanisms for analysis. These should be aimed for a study at *different levels of detail*. Indeed, given the enormous amount of biological data gathered in the last years, it is now a growing necessity for researchers to give such data a coherent meaning; the interest is then to identify and understand biological functions building on the available knowledge on basic elements such as proteins and genes. This requires following a system-level approach where isolated data is structured as to make up interactions that, in turn, will constitute more complex interactions at a higher level of abstraction.

The analysis at different levels of detail is also challenging from the semantic point of view because, depending on the level of detail, the *randomness* associated to the system (and to be captured by the semantics) varies. For instance, when the interest is on very fine descriptions of systems, individual reactions between components are described in terms of *rates*, and one usually deals with *stochastic simulations.* When the analysis is at a higher level of detail, deterministic models based on ordinary differential equations (ODEs) are usually a more convenient approach. This way, the need for analyzes at different levels of detail might call for the definition of several semantics for the language, each one of them providing different abstraction criteria over systems behavior. The several semantics not only should keep a certain consistency among them but should also respect the biological and physical conditions stipulated by the chosen level of analysis. This way, for instance, while a high level of abstraction could consider elementary *mass-action kinetic laws* (for which the rate of a single reaction is proportional to the product of the concentrations of the reactants), a lower level of abstraction could advocate the use of more precise criteria for systems description, such as *general kinetic laws* (which concern sequences of reactions). In order to be able to complement real experiments, the semantics should count with suitable reasoning techniques, that can be implemented as effective analysis tools within the frameworks biologists and other experts use.

In the second part of the paper, we shall give the rationale for a novel semantics for a discrete-timed concurrent constraint programming language. The semantics can be considered stochastic in that it considers rates during the evolution of the system. Nevertheless, seen as a whole, both language and semantics can capture different perspectives of the quantitative information associated to a (biochemical) system. This is not only useful for description purposes, but also appears promising for the definition of model-checking techniques for system analysis. More importantly, in the biological domain our semantics could adjust well to represent different levels of detail, by exploiting rates in conjunction with partial information in terms of constraints. We elaborate further on these ideas below.

Concurrent constraint programming (CCP) [1,2] is a declarative model for concurrency with strong ties to logic. In CCP, systems are described by pieces of partial information called *constraints*. Processes interact in a shared *store*; they either add new constraints or synchronize on the already available information. Notably, processes in CCP can be seen, at the same time, as computing agents and logic formulas. This not only constitutes a rather elegant approach for verification; it is fair to say that CCP provides a unified framework for system analysis.

In CCP, a great deal of quantitative information is captured by the notion of *constraint system*, a structure that defines logic inference capabilities over constraints. Constraint systems are parametric to CCP: several kinds of conditions (over integers, reals, strings) can be stated by choosing the appropriate constraint system(s). Timed concurrent constraint programming (`tcc`) [3] is a CCP-based framework for reactive systems. In `tcc`, time is explicitly represented as discrete time units in which computation takes place; `tcc` provides constructs to control process execution along such units. In previous works, `tcc` has been shown to have a place in systems biology [4,5].

In the light of stochastic models for quantitative information, the explicit time in `tcc` poses a legitimate question, that of determining to what extent the notions of stochastic duration and of discrete time unit can be harmoniously conciliated within a CCP-based framework. The question is relevant because it can give clues on clean semantic foundations for quantitative information in CCP, which in turn, should contribute to the development of more effective reasoning techniques over reactive systems in many emerging applications.

We outline preliminary results on an operational semantics for a `tcc` language with explicit stochastic durations, as presented in [6]. The proposed semantics aims at an explicit account of stochastically derived events using the description power of timed CCP calculi. This is a feature that in other CCP calculi (e.g. [7]) is handled at best implicitly. We define stochastic events in terms of the time units provided by the calculus: this provides great flexibility for modeling and, as mentioned before, it allows for a clean semantics. Most importantly, by considering stochastic information and adhering to explicit discrete time, it is possible to reason about processes using *quantitative* logics (both discrete and continuous), while retaining the simplicity of calculi such as `ntcc` [8] for deriving *qualitative* reasoning techniques (such as denotational semantics and proof systems). We consider existing qualitative reasoning techniques have a great potential for guiding/complementing the use of (usually costly) quantitative ones. Such an approach for applying qualitative techniques has shown to be useful in the biological context [9].

This work is part of a larger research program aimed at developing robust CCP-based techniques for analyzing complex applications and systems in computer music, security and biology. As such, it is our objective to formalize stochastic information in `tcc` in such a way that resulting languages and techniques (i) remain generic enough so to fit well in the target applications, and (ii) be amenable to efficient implementations, in the form of e.g. simulators and model-checkers.

Structure of this document. Next, we will give a (non exhaustive) overview of some languages for concurrency proposed for biological applications. In Section 3 we introduce our proposal for a stochastic semantics in the CCP model. An example of our approach is discussed in Section 4. Finally, we offer some concluding remarks (Section 5).

2 Some Languages for Concurrency in the Biological Setting

Languages for concurrency have been originally proposed for the description and analysis of concurrent and distributed systems. Main representatives include Milner's CCS [10] and Hoare's CSP [11], both proposed in the late 80s. In particular, CCS advocates for a compositional definition of concurrent systems, in which communication based on synchronization on designated *names* or *channels.* The π-calculus [12], one of the most successful languages in this research strand, builds on the communication scheme of CCS by allowing the use of channels as parameters in synchronizations, thus permitting the representation of systems with dynamic communication topologies.[4]

The first work exploiting languages for concurrency in the biological setting was the one by Regev, Silverman and Shapiro, who used the π-calculus to define the so-called *molecules as processes* abstraction [13]. Such an abstraction relates individual molecules with concurrent processes, defines molecular interaction as synchronous communication among processes, and reflects biochemical changes as transitions or mobility (which, in the π-calculus, is formalized as channel passing). Soon after this seminal work, the need for stochastic behavior in models was detected, and then Priami's *stochastic* π-calculus [14] was then adopted as target language, and tools based on it were proposed for the simulation of biochemical systems.

After the work of Regev et al., many other languages and formalisms from concurrency theory have been proposed for the study of biological phenomena, and one finds a myriad of motivations in each of them. Now we review three such formalisms; more in-depth treatments can be found elsewhere (see, for instance, [15] or the recent volume [16]).

2.1 PEPA

PEPA is a process calculi originally proposed for performance evaluation of systems [17]. In the biological context, PEPA has been used in the analysis of biochemical networks. Modeling in PEPA builds on the seminal work by Regev et al. by proposing a more abstract approach: instead of enforcing the analogy of *molecules as processes*, PEPA models aim at representing *species as processes*, taking into account the concentration of species rather than the actual number of molecules in them. This is justified by the high costs of carrying out numerical solutions for systems represented from a individual point of view. The syntax of PEPA includes the following kinds of operators:

- *Prefixes*, of the form $(\alpha, r).P$ which denote a process that has an action of type α. The duration of α is exponentially distributed with parameter r. Once this action is performed, the process behaves as P.

[4] The point-to-point communication scheme of CCS and the π-calculus could be arguably considered in opposition to the communication in CCP, which resembles more to a "broadcasting" scheme.

- *Choices*, of the form $P + Q$ which represent a system that might perform the actions of either P or Q. The first action that completes determines which one of the summands is preserved; the other is discarded.
- *Constant definitions*, of the form $C \overset{def}{=} P$, useful to assign names to patterns of behavior (here denoted as P).
- *Hiding*, denoted $P/\mathcal{H}$, defining a set of actions $\mathcal{H}$ that should remain internal or private to a process P.
- *Cooperation*, which defines the synchronization of two processes on a given a set of actions. Synchronization can be *multiway*, in the sense that several processes can synchronize on the same action; by having more than two processes synchronizing on some action, the rate associated with the synchronized activity should change.

At this point, a few words on the relation between constructs in PEPA with other languages for concurrency is worthwhile. The first five operands can be well considered as standard in languages for concurrency, and appear —in a way or another— in process calculi such as CCS and CSP. In PEPA, the cooperation operator could be regarded as a quantitative version of the traditional operator for *parallel composition*, which defines the concurrent execution of two different components. This composition preserves the actions of each of the components, and leads to synchronization in the case they perform some "compatible" action. In CCS, for instance, for each action l there is a *co-action* $\bar{l}$ which is meant to represent a behavior that is complementary to that of l; synchronization of processes exercising complementary actions is then labeled with τ, a special action symbol denoting *internal activity.*

In PEPA, the above operators allow for the definition of *abstract models* of biological networks, which can be analyzed from three different perspectives. First, it is possible to extract ODEs corresponding to the *population* perspective. Second. it is possible to extract a Continuous Time Markov Chain (CTMC) in which concentration amounts are discretized into suitable *levels* to facilitate numerical solutions. This is the *abstract* perspective. Finally, in the *individual* perspective, the CTMC is further specified so carry out finer analysis, based on the stochastic simulation.

One drawback of this original definition of PEPA is that interactions are binary, as in the π-calculus. This poses problems when representing more complex forms of synchronization that arise when more than two components are involved. Related with this, another downside of PEPA is that the set of kinetic laws representable in PEPA is also limited (mass-action kinetics). Based on these shortcomings, BioPEPA, a version of PEPA specifically tailored for the biological context, has been proposed [18]. While retaining the main features of the original framework, BioPEPA extends PEPA with functional rates, which allows to represent general kinetic laws.

BioPEPA allows for model checking techniques by means of PRISM [19], a probabilistic model checker. From the abstract model in BioPEPA it is possible to extract the CTMC required by PRISM to carry out experiments based on the Continuous Stochastic Logic (CSL).

2.2 BIOCHAM

BIOCHAM is a software environment for the description of biochemical systems. It provides description, verification and machine learning capabilities over systems. As for *description*, systems in BIOCHAM are represented by means of a simple notation based on (named) molecules, complexes (sets of molecules), modified forms of molecules (e.g. phosphorylated ones), and genes. Notice how this is a much simpler and concise language than the one described for PEPA (and BioPEPA). The dynamic aspect of systems is represented via a rule based language for expressing reaction rules. Remarkably, BIOCHAM allows for the specification of *rule patterns*, i.e. rules which involve *variables*, and that are capable to express diverse *rule instances*. This is useful for, e.g., experimenting with a system by varying some fixed set of parameters. Variables in rule patterns can be constrained with a simple set of constraints.

BIOCHAM offers a *multi-semantic* approach, in which models can be used to perform boolean, continuous or stochastic analyzes. This is done based on a simple Kripke semantics that provides a basis for comparing models, importing features from models in different formalisms and developing automated reasoning tools. For instance, in the *boolean* semantics one has a Kripke structure in which states are given by the presence/absence of the different biochemical entities of interest, while the transition relation is given by the defined reaction rules. Each one of these analyzes —intended for the study of the system at different levels of detail— is supported by a different semantic foundation.

A distinctive aspect of BIOCHAM is that it allows for the description of biological properties as temporal specifications. These complement rule-based representations by capturing the conditions and knowledge that are extracted from the experiments and should hold during various experiments and changes. One can *query* BIOCHAM specifications using Computation Tree Logic (CTL), and reason about reachability and stability queries, for instance. This is useful to formalize experimental knowledge as temporal specifications. Depending on the level of detail desired, one uses either CTL or a version of LTL with numerical constraints. BIOCHAM is connected to a well-established external model-checker (NuSMV) for performing relevant queries over a model. The *machine learning* capabilities of BIOCHAM are intended to automatically learn reaction rules. In fact, with the aid of techniques from inductive logic programming, and given a set of counterexamples and accessibility properties, this approach allows to discover missing reaction rules exploiting available temporal properties.

2.3 Untimed Stochastic Concurrent Constraint Programming

sCCP [7] is an untimed stochastic variant of CCP. This is obtained by adding an exponentially distributed stochastic duration to all the instructions interacting with the store. The rates are defined by functions mapping the current configuration of the store into a real number. This makes duration of processes explicitly dependent on context. The language is equipped with two transition relations: an instantaneous and a stochastic one. The instantaneous transition is made finite and confluent by imposing suitable syntactic conditions, hence the evolution of the system is given in terms of the stochastic relation. The underlying stochastic model can be both a CTMC or a Discrete Time Markov Chain (DTMC). There is an interpreter for the language in Prolog, exploiting the constraint logic programming libraries of SICStus Prolog in order to manage the constraint store. The interpreter's engine is based on the Gillespie's algorithm, and it allows to perform stochastic simulations of the models.

An sCCP program consists of a list of procedure declarations and of the starting configuration. Procedures are declared by specifying their name and their free variables, treated as formal parameters, procedure calls are to be guarded in order to avoid instantaneous infinite recursive loops. In sCCP, only *tell* and *ask* actions have a temporal duration. A tell operation includes a constraint into the store, whereas an ask is the basic means for querying information about the store by using the inference capabilities associated with, both actions are performed according to their duration. These actions can be combined together into a guarded mixed choice M. Procedure body consists of a sequence of agents, where any agent can choose stochastically an action between several ones (M) or it can perform an instantaneous tell or it can declare a variable local or it can be combined in parallel with other agents.

sCCP has been used to model two main kinds of biological systems: biochemical reactions and gene regulation networks. In these cases, variables will represent quantities that vary over time, e.g. the number of molecules of certain chemical species. Since sCCP is an untimed calculus and the variables are rigid, i.e., once instantiated their value is kept forever, the variables are modeled as growing lists with an unbounded tail. However, the lack of explicit time seems to obscure the use of reasoning capabilities similar to the ones used in other CCP-based languages like `ntcc` [8]. It is not clear how to analyze formally properties directly from a process as there is no an established relationship between a logic and the calculus. Notice also that the treatment of variables as lists makes more complex the representation of logical constraints over variables and variations of their value over time.

The formal verification is limited to the use of the underlying stochastic models as the logic nature of CCP is not exploited. sCCP allows for model checking techniques by means of the probabilistic model checker PRISM. A translation procedure associates to each sCCP program a (stochastically) equivalent one written in the language of PRISM.

3 Stochastic Behavior and Explicit Discrete Time in Concurrent Constraint Programming

We introduce `stcc`, a variant of `tcc` in which certain processes are annotated with a function λ, which represents the stochastic information in the language (see below). Annotated processes are tell, when and unless. With a slight abuse of notation, in tell and unless processes λ also stands for the constant value 1.We annotate unless as we see it as a counterpart of when processes. A careful definition of unless in the stochastic context, however, is yet to be completely determined. We do not discard that different applications (e.g. biological systems and computer music) need different unless definitions.

$$\begin{aligned} P, Q ::= \ & \mathsf{tell}_{\lambda}(c) \;\Big|\; \sum_{i \in I} \mathsf{when}\ c_i\ \mathsf{do}\ (P_i, \lambda_i) \;\Big|\; P \parallel Q \;\Big|\; \mathsf{local}\ x\ \mathsf{in}\ P \;\Big| \\ & !P \;\Big|\; \mathsf{next}\ (P) \;\Big|\; \mathsf{unless}_{\lambda}\ c\ \mathsf{next}\ (P) \end{aligned}$$

Operational Semantics. We use the same notion of discrete time as in `ntcc` and `tcc`. We assume that there are discrete time units of uniform size, each of them having its own constraint store. At each time unit, some stimuli are received from the environment; the process then executes with such stimuli as input. At the end of the time unit, some output is produced in the form of responses to the environment, and a residual process to be executed in the next time unit is scheduled. Information does not automatically transfer from one time unit to the following.

The operational semantics, given in Table 1, is defined over process-store configurations. We use γ, γ' to range over configurations, and assume a structural congruence relation $\equiv$ to identify processes with minor syntactic differences. The rules of the semantics carry both a *probability value* (denoted p) and a global *rate value* (denoted r). They decree two kinds of process execution, *immediate* (probability value equal to 1 and rate value `max`), and *stochastic*. In this sense, processes can be either immediate or stochastic. The idea of the semantics is to schedule immediate processes first, and then move to stochastic processes, whose execution involves a certain duration.

Rules for immediate execution resemble analogous rules in `tcc` and `ntcc`. The rule IMMTELL adds a constraint to the store as soon as possible. The rule IMMREP specifies that process $!P$ produces a copy P at the current time unit and then persists in the next time unit. There is no risk of infinite behavior within a time unit. In the Rule IMMUNLESS, process P is precluded if c is entailed by the current store d. The rule IMMINT allows for compositional extension.

Rules for stochastic executions consider the aforementioned function λ. Using the current store as parameter, λ describes how the global rate of the whole process varies. We use $\delta^m(P)$ to denote a *delay process* P with duration m: P will be executed at the m-th time unit from the current one. Given probability and rate values for a process, function $\varDelta$ determines its duration. The duration can be thus seen as an exponentially distributed random variable that depends on a probability and a rate.

The rule STOTELL defines stochastic tell actions. The rule STOCHOICE defines a choice over a number of guarded processes. Only those *enabled* processes, i.e., those whose guards entail from the current store, are considered. The rule STOINT defines the simultaneous occurrence of stochastic actions. As usual, the probability value is calculated assuming independence of the actions. Notice that the current store is not affected by stochastic actions; their influence is only noticeable in the following time units. The rules STOUNLESS and STOREP define unless and stochastic replicated actions, resp. The rule NEXT extends stochastic actions to next processes. In the rule LOCAL, local in P behaves like P, except that all the information on x produced by P can only be seen by P and the information on x produced by other processes cannot be seen by P. Notation $(\mathsf{local}\, x, c)\, P$ expresses that c is the local information produced by process $\mathsf{local}\, x$ in P. The rule STRCONG is self-explanatory.

These rules define behavior within a time unit; internal behavior takes place until reaching a configuration where no further computation is possible (*quiescence*). We need to define the *residual process* to be executed in the following time unit. We start by conjecturing that each quiescent configuration γ has a "standard" form:

$$\gamma \equiv \langle \prod_{j \in J} \mathsf{next}\,(P_j) \parallel \prod_{k \in K} \mathsf{unless}_1\, c_k\ \mathsf{next}\,(Q_k) \parallel \prod_{i \in I} \delta^{m_i}(P_i) \parallel \prod_{m \in M} \sum_{l \in I_m} \mathsf{when}\, c_l\ \mathsf{do}\,(P_l, \lambda_l),\ d \rangle.$$

$$\textsc{ImmTell}\ \frac{}{\langle \mathsf{tell}_1(d), c\rangle \longrightarrow_{1,\mathtt{max}} \langle \mathsf{skip}, c \wedge d\rangle} \qquad \textsc{ImmRep}\ \frac{\langle P, c\rangle \longrightarrow_{1,\mathtt{max}} \langle P', c'\rangle}{\langle !\, P, c\rangle \longrightarrow_{1,\mathtt{max}} \langle P \parallel \mathsf{next}\ (!\, P), c'\rangle}$$

$$\textsc{ImmUnless}\ \frac{}{\langle \mathsf{unless}_1\ c\ \mathsf{next}\ (P), d\rangle \longrightarrow_{1,\mathtt{max}} \langle \mathsf{skip}, d\rangle}\ \text{if } d \models c \qquad \textsc{ImmInt}\ \frac{\langle P, c\rangle \longrightarrow_{1,\mathtt{max}} \langle P', c'\rangle}{\langle P \parallel Q, c\rangle \longrightarrow_{1,\mathtt{max}} \langle P' \parallel Q, c'\rangle}$$

$$\textsc{StoTell}\ \frac{}{\langle \mathsf{tell}_\lambda(d), c\rangle \longrightarrow_{1,\lambda(c)} \langle \delta^m(\mathsf{tell}_1(d)), c\rangle}\ \text{with } m = \Delta(1, \lambda(c))$$

$$\textsc{StoChoice}\ \frac{}{\langle \sum_{i\in I} \mathsf{when}\ c_i\ \mathsf{do}\ (P_i, \lambda_i), c\rangle \longrightarrow_{p,r} \langle \delta^m(P_j), c\rangle}\ \text{if } c \models c_j$$
$$\text{with } r = \textstyle\sum_{i\in\{j \mid c\models c_j\}} \lambda_i(c);\ p = \lambda_j(c)/r;\ m = \Delta(p, r).$$

$$\textsc{StoInt}\ \frac{\langle P, c\rangle \longrightarrow_{p_1,r_1} \langle P', c\rangle \quad \langle Q, c\rangle \longrightarrow_{p_2,r_2} \langle Q', c\rangle}{\langle P \parallel Q, c\rangle \longrightarrow_{p',r'} \langle P' \parallel Q', c\rangle}\ \text{with } p' = p_1 \times p_2;\ r' = r_1 + r_2.$$

$$\textsc{StoUnless}\ \frac{}{\langle \mathsf{unless}_\lambda\ c\ \mathsf{next}\ (P), d\rangle \longrightarrow_{p,r} \langle \delta^m(\mathsf{unless}_1\ c\ \mathsf{next}\ (P)), d\rangle}\ \text{with } m = \Delta(p, r).$$

$$\textsc{StoRep}\ \frac{\langle P, c\rangle \longrightarrow_{p,r} \langle \delta^m(P), c'\rangle}{\langle !\, P, c\rangle \longrightarrow_{p,r} \langle \delta^m(P) \parallel \mathsf{next}\ (!\, P), c'\rangle} \qquad \textsc{Next}\ \frac{\langle P, c\rangle \longrightarrow_{p,r} \langle P', c\rangle}{\langle P \parallel \mathsf{next}\ (Q), c\rangle \longrightarrow_{p,r} \langle P' \parallel \mathsf{next}\ (Q), c\rangle}$$

$$\textsc{Local}\ \frac{\langle P, c \wedge \exists_x d\rangle \longrightarrow_{p,r} \langle P', c'\rangle}{\langle (\mathsf{local}\ x, c)P, d\rangle \longrightarrow_{p,r} \langle (\mathsf{local}\ x, c)P', d \wedge \exists_x c'\rangle} \qquad \textsc{StrCong}\ \frac{\gamma_1 \longrightarrow_{p,r} \gamma_2}{\gamma_1' \longrightarrow_{p,r} \gamma_2'}\ \text{if} \gamma_i \equiv \gamma_i'\ (i \in \{1, 2\})$$

Table 1. Operational semantics: internal transition rules.

In the form above, the first three components represent processes which can perform further only in the following time units. These are in contrast to the last component, i.e. $\prod_{m\in M} \sum_{l\in I_m} \mathsf{when}\ c_l\ \mathsf{do}\ (P_l, \lambda_l)$, which represent all the summations that did not trigger activity within the current time unit and that will be removed in the next one.

In the following definition we use A to denote the set of delayed processes in a quiescent configuration.

Definition 1 (Future function) *Given a quiescent configuration γ, its residual process is given by function F:*

$$F(\gamma) = \prod_{j\in J} P_j \parallel \prod_{k\in K} Q_k \parallel F'(A)$$

where function F' is defined as

$$F'(\delta^{m_1}(P_1) \parallel \ldots \parallel \delta^{m_n}(P_n)) = G(\delta^{m_1}(P_1)) \parallel \ldots \parallel G(\delta^{m_n}(P_n))$$

and where G is defined as

$$G(\delta^m(P)) = \begin{cases} \delta^{m-1}(P) & \textit{if } m > 1 \\ P & \textit{if } m = 1. \end{cases}$$

Unlike other languages like the stochastic π-calculus [20] or sCCP [7], it is worth noticing that in our semantics stochastic actions can evolve simultaneously; there is no a predefined order for execution. This way, for instance, $\mathsf{tell}_{\lambda_1}(c_1) \parallel \mathsf{tell}_{\lambda_2}(c_2)$ evolves into $\delta^{m_1}(\mathsf{tell}(c_1)) \parallel \delta^{m_2}(\mathsf{tell}(c_2))$ and in the next unit time, the configuration is $\delta^{m_1-1}(\mathsf{tell}(c_1)) \parallel \delta^{m_2-1}(\mathsf{tell}(c_2))$ (assuming m_1,$m_2 > 0$). This allows to naturally represent the evolution of different components in parallel.

Discussion. Since variables in `tcc` are logic (i.e. they can be defined at most once in each time unit), a potential source of inconsistencies is the simultaneous execution of several stochastic actions involving the same variables. This could represent a limitation in modeling. Consider for instance the kind of systems in which it is required to deal with quantities of elements of a certain type (as in biological reactions). In such systems, variables could be part of several actions, which would represent the changes over the elements in consideration. An inconsistency caused by two actions simultaneously altering the value of the same variable is clearly an undesirable feature. Therefore, there is the need for enhancing the semantics with a mechanism that imposes some kind of order over those actions related with potential inconsistencies. This would also presuppose modifications over rules calculating duration of stochastic actions, as concurrent actions would be simulated in a specific order. The formal definition of such a consistency mechanism is part of ongoing work.

Also, we are currently exploring convenient ways of modeling recursive calls and cells (i.e. variable assignments). One option, following the rationale in `ntcc` [8], is to introduce some form of non-determinism. However, this is a delicate issue in the stochastic setting: in the presence of non-determinism confluence for the operational semantics does not seem to hold, and the extraction of underlying stochastic models such as CTMCs or DTMCs appears more difficult. A different option is to achieve a similar modeling using suitable combinations of parallel compositions and stochastic choices. At present we are working on different encodings in this direction.

4 Example

In this section we illustrate our approach to model biological systems. We take the cycle of Rho GTP-binding proteins in the context of phagocytosis as case study. We first give a short biological description of the system and then we propose a `stcc` model representing its behavior.

Biological Description. *Phagocytosis* is a form of endocytosis by which a cell engulfs micro-organisms, large edible particles and cellular debris. The Rho GTP-binding proteins play an important role also in phagocytosis. These proteins act on the membranes of cells as molecular switches in several subcellular activities. i.e., it can be reversibly shifted between two or more states, regulating a variety of cell function as actin organization, cell shape and cell adhesion. In particular, Rac and Cdc42 are Rho GTP-binding proteins participating in Phagocytosis. While Rac is responsible for the branching structure of the actin polymerisation, Cdc42 causes the actin to polymerise in a linear structure.

Roughly, the cycle of Rho GTP-binding proteins can be described by reflecting how their state changes; this should consider their current bound (position) and their interactions. Options for the positioning are at the GTP (i.e. the nucleotide Guanosine TriPhosphate) or the GDP (i.e. the nucleotide Guanosine DiPhosphate), whereas interactions can take place with either Guanine-Nucleotide Exchange factors (GEFs) or GTPase activating proteins (GAPs). The Rho GTP-binding protein can thus shift (reversely) to a state at position GDP (GTP) and/or interacting with GEFs (GAPs proteins) producing the corresponding protein complex. We model the transformations of these proteins by using a set of chemical reactions describing their behavior. For example, $R \rightarrow_{0.1} \; RT$ describes a shift of a protein (represented by R) into a state in which it is GTP bounded at rate 0.1, where RT represents the resulting protein complex.

More in details, Rho GTP-binding proteins can be perceived as transmitting an incoming signal further to some effector in a molecular module by cycling between inactive and active states, depending on being GDP or GTP bound, respectively. GDP/GTP cycling is regulated by GEFs that promote the GDP dissociation and GTP-binding, whereas GTPase-activating proteins (GAPs) have the opposite effect and stimulate the hydrolysis of Rho GTP into Rho GDP. In the active GTP-bound state, Rho proteins interact with and activate downstream effectors, e.g., to control actin polymerisation in the context of Fc receptor mediated phagocytosis [21].

Previously, in [22] it was proposed a computational model of the Rho GTP-binding proteins by means of ordinary differential equations. Chemical reactions underlying the ordinary differential equations can be obtained. We follow an approach similar to that one of [21] by modeling the different chemical reactions in our calculus.

4.1 A `stcc` model of the cycle of Rho GTP-binding proteins in the context of phagocytosis

Unlike [21], in our model, we can represent each one of the chemical reactions independently. Before entering into the detailed description of the model let us informally describe two encodings for recursive functions and mutable entities that will allow for cleaner model descriptions. Based on the discussion at the end of the previous section, in the models below we shall assume suitable encodings of some form of choices for modeling both recursive functions and mutable entities:

- We shall assume the encoding of *recursive definitions* of the form $q(x) =_{def} P_q$, where q is the process name and P_q calls q only once and such a call must be within the scope of a "next" or a delay process δ^m.
- *Cells*, a basis for the specification and analysis and persistent data structures, can be thought of as structure that contains a value, and if tested, it yields this value. A cell keeps its value over the time units until it is modified. We use notations $x : v$ and $x := v$ to represent the initialization and the assignment of a cell c with value v, respectively. Also, we shall use notation $x := x + z$ as an abbreviation of the assignment $x := x' + z$, where x' is the value of the cell x in the previous time unit and z is fixed value. The operation $x := x - z$ can be encoded analogously.

We now enter to describe the `stcc` model representing the cycle of Rho GTP-binding proteins in phagocytosis. The model assumes a constraint system over finite domains of integers. The model involves a series of persistent variables (modeled as cells) that store the number of each protein (complex) evolved into the chemical reactions describing the model. In general, a chemical reaction is of the form $R_1 + \ldots + R_n \rightarrow_k P_1 + \ldots + P_m$, where the n reactants $R_i's$ (in this model, the reactants are the proteins —or protein complexes— present at the initiation of the reaction) are transformed into the m products $P_j's$ (in this model, the products are the proteins —or protein complexes— resulting from the chemical reaction). Either n or m can be equal to zero; the case $m = 0$ represents a degradation reaction, while the case $n = 0$ represents an external feeding of the products. Each reaction has an associated rate value k, representing essentially its basic speed. It could be described as following:

$$
\begin{aligned}
\textsc{Reaction} =_{def} \ & \mathsf{unless}_1 \bigwedge_{i=1}^{n} (R_i > 0) \ \mathsf{next} \ (\textsc{Reaction}) \ \| \\
& \mathsf{when} \bigwedge_{i=1}^{n} (R_i > 0) \ \mathsf{do} \\
& (\prod_{i=1}^{n} R_i := R_i - 1 \ \| \ \prod_{i=1}^{m} P_i := P_1 + 1 \ \| \ \textsc{Reaction}, f(k))
\end{aligned}
$$

In the previous definition, it is checked if all the reactants are present in the system by verifying that the quantity of each one is greater than zero, these conditions can be seen as guards. If the guards are entailed from the current store, process REACTION modifies the quantity of reactants and products consistently, a recursive call is made to keep persistently the reaction as part of the model. The execution of the reaction is subjected to $f(k)$, the rate of the reaction, which indicates that different functions can be used to describe the rate of the reaction from a value k. If the reactants are not present in the system, then in the next time unit a recursive call is made. Notice that effectively only one recursive call is made either by while or unless constructor.

Now, we shall present the chemical reactions to be modeled:

- $R \rightarrow_{0.1} RT$, $RT \rightarrow_{0.02} R$, $R \rightarrow_{0.033} RD$, $RD \rightarrow_{0.02} R$, $RT \rightarrow_{0.02} RD$.
- $RA \rightarrow_{0.0085} RTA$, $RTA \rightarrow_{0.0002} RA$, $RA \rightarrow_{0.1} RDA$, $RDA \rightarrow_{0.02} RA$, $RTA \rightarrow_{2104} RDA$.
- $RE \rightarrow_{0.1} RTE$, $RDE \rightarrow_{0.033} RE$, $RTE \rightarrow_{0.02} RDE$.
- $RDA \rightarrow_{500} RD$, $RA \rightarrow_{500} R$, $RTE \rightarrow_{76.8} RT$, $RE \rightarrow_{0.43} R$.
- $RT + A \rightarrow_{3} RTA$, $R + A \rightarrow_{1} RA$, $R + E \rightarrow_{0.43} RE$, $RD + E \rightarrow_{0.0054} RDE$

Where R denotes the Rho GTP-binding protein, whereas RD and RT denote its GDP and GTP bound forms respectively. A and E denote GAP and GEF, respectively. Thus, RDE, denotes the protein complex formed by RD and E, RA denotes the complex formed by R and A, RTA denotes the complex formed by RT and A, RDA denotes the complex formed by RD and A, RE denotes the complex formed by R and E and RTE denotes the complex formed by RT and E.

Each reaction has an associated rate k, representing its basic speed. The actual rate of the reaction is $k \cdot R_1 \cdot \ldots \cdot R_n$, where R_i denotes the number of proteins (complex proteins) of type R_i present in the system.

As an example, reactions $RT + A \rightarrow_3 RTA$ and $R + A \rightarrow_1 RA$ can be modeled as following:

$$
\begin{aligned}
\textsc{Reaction}_1 =_{def}\ & \mathsf{unless}_1\ (RT > 0 \wedge A > 0)\ \mathsf{next}\ (\textsc{Reaction}_1) \parallel \\
& \mathsf{when}\ RT > 0 \wedge A > 0\ \mathsf{do} \\
& \quad (RT := RT - 1 \parallel A := A - 1 \parallel RTA := RTA + 1 \parallel \textsc{Reaction}_1, RT \cdot A \cdot 3) \\
\textsc{Reaction}_2 =_{def}\ & \mathsf{unless}_1\ (R > 0 \wedge A > 0)\ \mathsf{next}\ (\textsc{Reaction}_2) \parallel \\
& \mathsf{when}\ R > 0 \wedge A > 0\ \mathsf{do} \\
& \quad (R := R - 1 \parallel A := A - 1 \parallel RA := RA + 1 \parallel \textsc{Reaction}_2, R \cdot A \cdot 1)
\end{aligned}
$$

A network with these two reactions can be modeled as the corresponding parallel composition $\textsc{Reaction}_1 \parallel \textsc{Reaction}_2$; this way we can model the whole system.

A more detailed model can capture the behavior of these proteins together with the effectors. The binding of E to the RT results in the formation of a complex protein consisting of RT, E and the effector protein, M denotes this complex. According to a simplified model, the following reactions describe —to some extent— the behavior.

– $RT + E \rightarrow_{600} M, M \rightarrow_{18} RT + E, RD + M \rightarrow_{0.6} RT + M.$

Clearly, our model can be extended just by putting in parallel the processes modeling the behavior of each one of the reactions above.

`stcc` offers two clear advantages in the modeling of this kind of systems w.r.t [21]. On the one hand, the calculus provides flexibility in the use of rates by allowing the definitions of arbitrary functions. In this way it is possible not only to model mass-action-like reactions but more complex expression for the rate of the reaction, e.g. Michaelis-Menten kinetics. On the other hand, we can extend the model in a fully compositional way, where the extensions do not bring modifications on the rest of the model. We think that a tool for `stcc` could assist in simulating experiments and models in a more intuitive way.

5 Concluding Remarks

We have proposed `stcc`, an extension to `tcc` aimed at handling stochastic information in the modeling and verification of reactive systems. Biological systems constitute an important part of the systems we are interested in. Besides the example presented above, our language and semantics could have other applications in the biological domain. This is supported by the fact that CCP-based calculi have shown to be convenient for modeling, simulating and verifying several kinds of biological systems [5,4,7]. Our approach allows for the presence of both partial quantitative information at the level of the constraint systems, the presence of functional rates and it can potentially makes use of reasoning techniques inherited from `ntcc` to prove that a given process P satisfy a given property F.

We also reviewed some languages for concurrency aimed for biological applications. One such languages is sCCP [7], which is perhaps the calculus more closely related to ours. In our view, sCCP has several drawbacks: it does not include an explicit notion of time and does not exploit the logic nature of CCP for verification. Also, sCCP lacks a means of expressing absence of information, which has proven most useful in the biological context [4]. The explicitly timed CCP language `ntcc` [8] provides both a proof system and a means of representing absence of information. In fact, `ntcc` was used in [5,4] to model different biological systems using two kinds of partial information: *behavioral* (e.g. the unknown relative speeds on which a system evolves) and *quantitative* (e.g. the set of possible values that a variable can take). It must be noticed that `ntcc` does not allow for stochastic or probabilistic information.

Based on the above, we think that the extension to `tcc` here proposed could serve several purposes in the biological context. The most immediate use is the definition of enhanced models of systems already modeled in `ntcc` (the Sodium-Potassium pump, regulation and mutation processes in genetic regulatory networks). Also, although it is not evident that every sCCP process can be translated into our language (the tell operator in sCCP has continuation), we are confident we can model most of the biological systems described in [7].

References

1. Saraswat, V.: Concurrent Constraint Programming. The MIT Press, Cambridge, MA (1993)
2. Olarte, C., Rueda, C., Valencia, F.: Concurrent Constraint Programming: Calculi, Languages, and Emerging Applications. Newsletter of the ALP **21**(2-3) (2008)
3. Saraswat, V.A., Jagadeesan, R., Gupta, V.: Foundations of timed concurrent constraint programming. In: LICS, IEEE Computer Society (1994) 71–80
4. Arbeláez, A., Gutiérrez, J., Pérez, J.A.: Timed Concurrent Constraint Programming in Systems Biology. Newsletter of the ALP **19**(4) (2006)
5. Gutiérrez, J., Pérez, J.A., Rueda, C., Valencia, F.D.: Timed concurrent constraint programming for analysing biological systems. Electr. Notes Theor. Comput. Sci. **171**(2) (2007) 117–137
6. Aranda, J., Pérez, J.A., Rueda, C., Valencia, F.: Stochastic behavior and explicit discrete time in concurrent constraint programming. In: Proc. of ICLP 2008. Volume 5366 of LNCS., Springer (2008)
7. Bortolussi, L.: Constraint-based approaches to stochastic dynamics of biological systems. PhD thesis, University of Udine (2007)
8. Nielsen, M., Palamidessi, C., Valencia, F.D.: Temporal concurrent constraint programming: Denotation, logic and applications. Nord. J. Comput. **9**(1) (2002) 145–188
9. Fages, F., Soliman, S.: Formal cell biology in biocham. In: SFM. Volume 5016 of LNCS., Springer (2008) 54–80
10. Milner, R.: Comunication and Concurrency. International Series in Computer Science. Prentice Hall (1989)
11. Hoare, C.A.R.: Comunicating Sequential Processes. International Series in Computer Science. Prentice Hall (1985)
12. Sangiorgi, D., Walker, D.: The π-calculus: a Theory of Mobile Processes. Cambridge University Press (2001)
13. Regev, A., Silverman, W., Shapiro, E.Y.: Representation and simulation of biochemical processes using the pi-calculus process algebra. In: Pacific Symposium on Biocomputing. (2001) 459–470
14. Priami, C.: Stochastic π-calculus. The Computer Journal **38**(6) (1995) 578–589

15. Gilmore, S., Hillston, J.: Performance evaluation comes to life: quantitative methods applied to biological systems. SIGMETRICS Performance Evaluation Review **35**(4) (2008) 3–13
16. Bernardo, M., Degano, P., Zavattaro, G., eds.: Formal Methods for Computational Systems Biology, 8th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2008, Bertinoro, Italy, June 2-7, 2008, Advanced Lectures. In Bernardo, M., Degano, P., Zavattaro, G., eds.: SFM. Volume 5016 of Lecture Notes in Computer Science., Springer (2008)
17. Hillston, J.: A compositional approach to performance modelling. Cambridge University Press (1996)
18. Ciocchetta, F., Hillston, J.: Process algebras in systems biology. In: Proc. of SFM. Volume 5016 of Lecture Notes in Computer Science., Springer (2008) 265–312
19. Oxford University: PRISM Model Checker (2008) See `http://www.prismmodelchecker.org`.
20. Priami, C.: Stochastic pi-calculus. Comput. J. **38**(7) (1995) 578–589
21. Cardelli, L., Gardner, P., Kahramanogullari, O.: A process model of rho gtp-binding proteins in the context of phagocytosis. Electr. Notes Theor. Comput. Sci. **194**(3) (2008) 87–102
22. Goryachev, A., Pokhilko, A.: Computational model explains high activity and rapid cycling of rho gtpases within protein complexes. plos computational biology. PLOS Computational Biology, **2** (2006) 1511–1521