Stochastic Behavior and Explicit Discrete Time in Concurrent Constraint Programming*

Jesús Aranda[1,2], Jorge A. Pérez[3], Camilo Rueda[4,5], and Frank D. Valencia[6]

[1] INRIA Futurs and LIX, Ecole Polytechnique, France
[2] Escuela de Ingeniería de Sistemas y Computación, Universidad del Valle, Colombia
[3] Dept. of Computer Science, University of Bologna, Italy
[4] Dept. of Science and Engineering of Computing, Universidad Javeriana - Cali, Colombia
[5] IRCAM, Paris, France
[6] CNRS and LIX, Ecole Polytechnique, France

Abstract. We address the inclusion of stochastic information into an *explicitly timed* concurrent constraint process language. An operational semantics is proposed as a preliminary result. Our approach finds applications in biology, among other areas.

Motivation. 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.

Concurrent constraint programming (CCP) [1] 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. Timed concurrent constraint programming (`tcc`) [2] is a declarative 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 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

* Research partially supported by the COLCIENCIAS project REACT (No. 1251-330-18902) and the INRIA 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).

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. In this paper, we outline preliminary results on an operational semantics for a tcc language with explicit stochastic durations.

More into details, 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. [3]) 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 [4] 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 [5].

This work is part of a larger research programme 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.

Description. We consider 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.

$$P, Q ::= \mathsf{tell}_\lambda(c) \;\Big|\; \mathsf{when}\, c\, \mathsf{do}\, (P, \lambda) \;\Big|\; P \parallel Q \;\Big|\; \mathsf{local}\, x\, \mathsf{in}\, P \;\Big|\; !P \;\Big|\; \mathsf{next}\, (P) \;\Big|\; \mathsf{unless}_\lambda\, c\, \mathsf{next}\, (P)$$

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 Δ 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 $\mathsf{local}\, x\, \mathsf{in}\, P$. 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:

$$\gamma \equiv \langle \prod_{j \in J} \mathsf{next}\ (P_j) \parallel \prod_{k \in K} \mathsf{unless}\ c_k\ \mathsf{next}\ (Q_k) \parallel \prod_{i \in I} \delta^{m_i}(P_i),\ d \rangle.$$

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:*

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

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

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

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

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

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

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

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

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

Table 1. Operational semantics: internal transition rules.

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 [6] or sCCP [3], 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.

Applications in Biology. We think that our language and semantics have applications in the biological domain. This is supported by the fact that CCP-based calculi have shown to be convenient for modelling, simulating and verifying several kinds of biological systems [7,8,3]. In [3], stochastic concurrent constraint programming (sCCP) is used to model biochemical reactions and gene regulatory networks. Functional rates in sCCP give considerable flexibility to formulate reactions. However, sCCP 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 [8]. The explicitly timed ccp language `ntcc` [4] provides both a proof system and a means of representing absence of information. In fact, `ntcc` was used in [7,8] 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 [3]. We also plan to analyse the model in [9], which describes the cycle of Rho GTP-binding proteins in the context of phagocytosis.

References

1. Saraswat, V.: Concurrent Constraint Programming. The MIT Press, Cambridge, MA (1993)
2. Saraswat, V.A., Jagadeesan, R., Gupta, V.: Foundations of timed concurrent constraint programming. In: LICS, IEEE Computer Society (1994) 71–80
3. Bortolussi, L.: Constraint-based approaches to stochastic dynamics of biological systems. PhD thesis, University of Udine (2007)
4. Nielsen, M., Palamidessi, C., Valencia, F.D.: Temporal concurrent constraint programming: Denotation, logic and applications. Nord. J. Comput. **9**(1) (2002) 145–188
5. Fages, F., Soliman, S.: Formal cell biology in biocham. In: SFM. Volume 5016 of LNCS., Springer (2008) 54–80
6. Priami, C.: Stochastic pi-calculus. Comput. J. **38**(7) (1995) 578–589
7. 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
8. Arbeláez, A., Gutiérrez, J., Pérez, J.A.: Timed Concurrent Constraint Programming in Systems Biology. Newsletter of the ALP **19**(4) (2006)
9. 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