A Stochastic Non-deterministic Temporal Concurrent Constraint Calculus

Carlos Olarte
Pontificia Universidad Javeriana Cali
Dept. Ingeniería de Sistemas y Computación
Calle 18 No 118-250 Av. Cañasgordas, Cali, Colombia
caolarte@atlas.puj.edu.co

Camilo Rueda
crueda@atlas.puj.edu.co

Abstract

We propose **sntcc** *, a stochastic extension of the* **ntcc** *calculus, a model of* temporal concurrent constraint programming *with the capability of modeling asynchronous and non-deterministic timed behavior. We argue that such an extension is needed to faithfully model concurrent systems in real-life situations. We provide a suitable temporal logic and proof system for* **sntcc** *and illustrate how to use them for proving properties of stochastic systems. We argue that this modeling strategy of using explicit stochastic constructs within the calculus provides a "runnable specification" for a wide variety of stochastic systems that eases the task of formally reasoning about them. We give examples of specifications in* **sntcc** *and use the extended linear temporal logic for proving properties about them.*

1. Introduction

Concurrent constraint (cc) process calculi ([11]) provide formal grounds to the integration of concurrency and constraints so that non trivial properties of concurrent systems can be expressed and proved. We are interested in using concepts and techniques from concurrent processes modeling to define suitable computational calculi and analyze their behavior in real life settings. In particular, we propose using a temporal non deterministic concurrent calculus (**ntcc** , see [5]) as a formal base to model processes in such a way that their properties can be formally proved. However, in many real life situations interactions between processes occur according to stochastic rules. In biological systems, for example, the occurrence of some particular reactions is determined by issues such as affinities, distances, relative concentrations of components, etc. These kind of systems would be awkward to model in **ntcc** .

In this paper we propose **sntcc** , an extension of **ntcc** with constructs for expressing stochastic behavior. Since we want to express and prove properties of stochastic systems modeled with the calculus, we also extend the linear temporal logic and proof system associated with **ntcc** . What we gain from this *low level* approach in system modeling is twofold. One the one hand, we are able to ground the development of stochastic process simulation tools on a very precise formal foundation. On the other hand, our model can give us clues for constructing formal proofs of interesting properties of a given stochastic process. We give examples to illustrate the expressiveness of **sntcc** and the convenience of the extended temporal logic to prove properties involving probabilistic reasoning.

The main contributions of this paper are: 1) the definition of **sntcc** a stochastic concurrent constraint process calculus extending **ntcc** with constructs that explicitly assign probabilities to processes and 2) extending the temporal logic and proof system of **ntcc** in such a way that by modeling a stochastic process in **sntcc** one inherits a well defined logical inference system that can be used to prove its properties (or lack thereof).

2. NTCC Calculus

In concurrent constraint calculi such as **ntcc** , process interactions can be determined by partial information (i.e. constraints) accumulated in a global store. The particular type of constraints is not fixed but specified in a *constraint system* that is considered a parameter of the calculus.

2.1. Constraint System

A constraint represents a piece of partial information over a set of variables. For example, in constraint $x > 3$, the value of x is unknown but we can assert that it is greater than 3.

A constraint system provides a signature from which constraints can be constructed. It also provides an entailment relation ($\models$) over constraints where $c_1 \models c_2$ holds iff the information of c_2 can be inferred from c_1.

Formally, a constraint system is a tuple $\langle \sum, \Delta \rangle$ where $\sum$

is a signature (i.e a set of constants, functions and predicate symbols) and Δ is a consistent first-order theory over $\sum$ (i.e a set of sentences over $\sum$ having at least one model). Constraints can be viewed as first-order formulae over $\sum$ and $c \models d$ holds if the implication $c \Rightarrow d$ is valid in Δ [5]. For practical reasons the entailment relation must be decidable.

A constraint *store* is a set of variables and a conjunction of formulae (the constraints). The store is used by processes to share information and for synchronization purposes. The store is monotonically refined by adding information using $tell$ operations of the calculus. For example, $tell(x < 2)$ adds constraint $x < 2$ to the store. Additionally, it is possible to test if a constraint c can be entailed from the store by means of so-called ask operations. For example, $ask(x < 5)$ tests whether it is possible to infer that $x < 5$ from the information contained in the $store$ (i.e. $store \models x < 5$). The ask operation blocks when neither $store \models x < 5$ nor $store \models \neg(x < 5)$ holds.

2.2. ntcc overview

ntcc [5] is a process calculus that extends **tcc** [9]. In both of them, processes share a common store of partial information [11]. Both **ntcc** and **tcc** have an explicit notion of (discrete) time. In **ntcc** time is conceptually divided into *discrete intervals or time-units*. In a particular time interval, a deterministic concurrent constraint process receives a stimulus (a constraint) from the environment and it is executed with this stimulus as the initial store. When it reaches its resting point, it responds to the environment with the resulting store. The resting point also determines a residual process which is then executed in the next time interval. **ntcc** has been successfully used to model many real life system such as reactive systems, robot behavior and music composition.

Unlike **tcc** , **ntcc** includes constructs for modeling *nondeterminism* and *asynchrony*. A very important benefit of being able to specify non-deterministic and asynchronous behavior arises when we are modeling the interaction among several components running in parallel, where each component is part of the environment of the others.

2.3. Process Syntax

Table 1 describe briefly the syntax of **ntcc** processes. See [5] for further details.

Two new operators will be introduced in section 3 to express stochastic behavior. Next, we will illustrate their use in an example in section 4.

Table 1. ntcc Process Syntax

Process	Description
$\mathbf{tell}\ c$	Adds constraint c to the store.
$\sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i$	Chooses non-deterministically a process P_i whose guard c_i is entailed by the store.
$P \| \| Q$	Represents the parallel composition between P and Q.
$\mathbf{local}\ x\ \mathbf{in}\ P$	Behaves like P but the information of the variable x is local to P, i.e. P cannot see information about a global variable x and processes which are not part of P cannot see the information generated by P about x.
$\mathbf{next}\ P$	Executes process P in the next time unit (unit-delay)
$\mathbf{unless}\ c\ \mathbf{next}\ P$	Executes P iff c cannot be entailed by the constraint store *in the current time unit*
$!P$	Executes P in all time units from the current one. It can be viewed as $P \| \| \mathbf{next}\ P \| \| \mathbf{next}\ \mathbf{next}\ P \| \| ...$
$\star P$	Represents unbounded but finite delays, i.e P eventually will be executed. This process can be viewed as $P + \mathbf{next}\ P + \mathbf{next}\ \mathbf{next}\ P ... \mathbf{next}^{\,n} P$ where n is a finite natural number.

2.4. Internal reductions

In this section we show the **ntcc** operational semantics by giving reduction rules for each constructor. These rules will help us to understand how **ntcc** processes interact with each other until they reach a resting point. Recall that when this state is reached, another time units is created with an empty constraint store and the *residual* process is executed. For a complete description of **ntcc** semantics refer to [5]. Reduction rules are based on *configurations*. A configuration $\langle P, s \rangle$ is composed of a **ntcc** process P and a store s.

For $\mathbf{tell}$ processes we have:

$$TELL \frac{}{\langle \mathbf{tell}\ c, d \rangle \rightarrow \langle \mathbf{skip}, d \wedge c \rangle}$$

where $\mathbf{skip}$ is the empty process. This reaction says that a $\mathbf{tell}$ process adds information (a constraint) to the constraint store d.

Let I a finite set of indexes and $i, j \in I$. The system reacts as follows:

$$SUM \frac{}{\langle \sum_{i \in I} \textbf{when } c_i \textbf{ do } P_i, d\rangle \rightarrow \langle P_j, d\rangle}$$

Iff $d \models c_j$. It means that a particular process P_j is non-deterministically chosen for execution among all those whose guard (c_i) can be entailed from the current store d.
For parallel composition we have:

$$PAR \frac{\langle P, c\rangle \rightarrow \langle P', d\rangle}{\langle P||Q, c\rangle \rightarrow \langle P'||Q, d\rangle}$$

It says that if P evolves to P', then the same transition can occur if we execute P in parallel with some process Q. Parallel composition is commutative.
In $!P$ (replication) processes the rule is:

$$REP \frac{}{\langle !P, d\rangle \rightarrow \langle P||\textbf{next } !P, d\rangle}$$

For $\textbf{unless } c \textbf{ next } P$ processes:

$$UNLESS \frac{}{\langle \textbf{unless } c \textbf{ next } P, d\rangle \rightarrow \langle \textbf{skip}, d\rangle} if \ \ d \models c$$

The rule says that nothing is done when c is entailed by the store.
Finally, the rule for star processes is:

$$STAR \frac{}{\langle \star P, d\rangle \rightarrow \langle \textbf{next }^n P, d\rangle} if \ \ n \geq 0$$

This models the fact that process P will be run in the (undetermined) future.

The above rules define so-called *internal* transitions. In addition to these, **ntcc** defines an *observable* transition which is the one that goes from one time unit to the next. At the end of a time unit the resulting store can be observed by the environment. Then, processes contained in **next** constructs are scheduled for the next time unit. These include those defined by **unless** processes whose guard cannot be entailed from the current store (see [5] for details).

2.5. Linear-temporal Logic in ntcc

ntcc can be used to verify properties over timed systems. It provides for this a linear temporal logic in which temporal properties over infinite sequences of constraints can be stated [5]. The syntax of this logic is as follows:

$$A, B, ... : c \mid A \dot{\Rightarrow} A \mid \dot{\neg} A \mid \dot{\exists}_x A \mid \ \circ A \mid \Diamond A \mid \Box A$$

c is a constraint. $\dot{\Rightarrow}$, $\dot{\neg}$ and $\dot{\exists}_x$ represent the linear-temporal logic implication, negation and existential quantification, respectively [4]. These symbols should not be confused with their counterpart in the constraint system (i.e $\Rightarrow$, $\neg$ and $\exists$). Symbols $\circ$, $\Box$ and $\Diamond$ denote the temporal operators $next$, $always$ and $eventually$.

The interpretation structures of formulae in this logic are infinite sequences of states [4]. In **ntcc** , states are replaced by constraints. Given the set C of constraints in the constraint system, let $\alpha \in C^\infty$ be an infinite sequence of constraint and $\alpha(i)$ the $i-th$ element of α. We say that $\alpha \in C^\infty$ is a model of (or that it satisfies) A, notation $\alpha \models A$, if $\langle \alpha \ , \ 1\rangle \models A$ where:

$$\begin{array}{lll}
\langle \alpha \ , \ i\rangle \models c & iff & \alpha(i) \models c \\
\langle \alpha \ , \ i\rangle \models \dot{\neg} A & iff & \langle \alpha \ , \ i\rangle \not\models \ A \\
\langle \alpha \ , \ i\rangle \models A_1 \dot{\Rightarrow} A_2 & iff & \langle \alpha \ , \ i\rangle \models A_1 \ implies \\
 & & \langle \alpha \ , \ i\rangle \models A_2 \\
\langle \alpha \ , \ i\rangle \models \circ A & iff & \langle \alpha \ , \ i+1\rangle \models A \\
\langle \alpha \ , \ i\rangle \models \Box A & iff & \forall_{j \geq i} \langle \alpha \ , \ j\rangle \models A \\
\langle \alpha \ , \ i\rangle \models \Diamond A & iff & \exists_{j \geq i} \ s.t. \langle \alpha \ , \ j\rangle \models A \\
\langle \alpha \ , \ i\rangle \models \dot{\exists}_x A & iff & there \ is \ an \ x-variant \\
 & & \alpha' \ of \ \alpha \ s.t. \ \langle \alpha' \ , \ i\rangle \models A
\end{array}$$

In the last expression, α' and d are $x-$variants of α and c, respectively, if they are the same except for the information about x.

In [5] a proof system is built on top of this logic. Given a process P and a formula A, a proof of $P \models A$ can be obtained by following a set of inference rules. In the next section we introduce two stochastic constructors on **ntcc** and next we modify this inference system to deal with probability statements like *"this process will be executed eventually under a probability ρ"* or *"Formula A can be entailed in some time unit with a given probability"*.

3. Introducing Stochastic Behavior in ntcc

In real life systems the notion of *uncertainty* or *probability* is ubiquitous. For example, if we want to model malfunctioning in some system (e.g an electronic device), each component may have a different probability to suffer some damage. We think that is necessary to introduce stochastic behavior in process calculus such as **ntcc** to be able to represent faithfully stochastic processes occurring in many areas. In [7] stochastic features are added to the π-calculus [8]. Basically, a real number is associated to each process communication channel. This number denotes how probable it is to establish communication through it. An external stochastic control orders the communication channels according to this value and chooses which reaction (handshaking between some name x and a co-name $\overline{x}$) must happen.

Nevertheless, calculi such as **ntcc** or **tcc** are more suitable to describe real life systems in a declarative way due to its underlying constraint system. In the previous example about the electronic device, we can know if there is a damage in the global system in a straightforward manner by *asking* to the constraint store if a suitable predicate , say $comp_x\_fail$, can be entailed from it.
Besides of [7], there are other probabilistic extensions of process calculus. In particular, [2] proposes a probabilistic extension for cc ([10]) and **tcc** ([9]) leading to a synchronous reactive probabilistic programming language called **pcc** and **timed pcc** respectively. In this section we define **sntcc** , a stochastic non-deterministic concurrent constraint calculus derived from **ntcc** . Unlike **timed pcc**, **sntcc** supports asynchrony and non-deterministic behavior (features inherited from **ntcc**) and additionally, it provides a linear temporal logic and an inference system to prove properties over systems modeled on it. In section 4 we argue about the expressiveness of this new language and prove some properties using the inference system proposed.

3.1. An eventual-stochastic Construct ($\star_\rho P$)

As seen in section 2.4 eventuality can be modeled in **ntcc** by using $\star$ constructors. Following our example, assume that our electronic device is composed of two distinct parts and that any one of them may eventually fail. In **ntcc** we can model this fact by executing a process representing the system in parallel with two star processes as follows:

$$\begin{aligned} P_1 &\stackrel{def}{\equiv} \star(\mathbf{tell}\ Comp_1\_Fail) \\ P_2 &\stackrel{def}{\equiv} \star(\mathbf{tell}\ Comp_2\_Fail) \\ \mathbf{DEVICE} &\stackrel{def}{\equiv} SYSTEM||P_1||P_2 \end{aligned} \tag{1}$$

If part P_1 has a bigger chances to fail than part P_2 we should be able to observe this behavior. For this reason we propose a new constructor $\star_\rho P$ for **ntcc** with the following operational semantics:

$$STAR_\rho \frac{}{\langle \star_\rho P, d\rangle \rightarrow \langle \mathbf{next}^n P, d\rangle} if\ n \geq 0 \wedge \Phi(\rho) = 1$$

Intuitively, $\Phi(\rho) : \Re \in [0,1] \rightarrow bool$ denotes a function that given a probability ρ returns a boolean value that can be used to decide whether a process should or should not be run. This function can be computed by generating pseudo-random numbers following some probabilistic distribution, e.g a binomial distribution with 1 as number of events. Thus, $\star_\rho P$ represents the process that will eventually execute P under a probability ρ.
$\star_\rho$ processes can in principle be expressed using standard constructors of **ntcc** by adding the function Φ to the signature of the constraint system:

$$\star_\rho P \stackrel{def}{\equiv} \mathbf{local}\, x\ \mathbf{in}\ (tell(x = \Phi(\rho))||\mathbf{when}\ x = 1\ \mathbf{do}\ \star P)$$

Notice that in

$$\star_{\rho_1}(\mathbf{tell}\ Comp_1\_Fail)||\star_{\rho_2}(\mathbf{tell}\ Comp_2\_Fail)$$

failures in component one will be observed more frequently than failure in component two if $\rho_1 \geq \rho_2$. On the other hand, no failure will be observed if $\Phi(\rho_1) = \Phi(\rho_2) = 0$.

3.2. Stochastic-process Construct ($_\rho P$)

The $\star_\rho$ process defined above operate over an arbitrary time interval. As we mentioned in the last example, the process may end up not been executed at all, depending on $\Phi(\rho)$. To represent processes that are executed in the current time unit under certain probability we propose construct $_\rho P$. Operationally:

$$RHOP \frac{}{\langle {}_\rho P, d\rangle \rightarrow \langle P, d\rangle} if\ \Phi(\rho) = 1$$

This kind of process can be expressed as follows:

$${}_\rho P \equiv \mathbf{local}\ x\ \mathbf{in}\ \mathbf{tell}\ (x = \Phi(\rho))||\mathbf{when}\ x = 1\ \mathbf{do}\ P \tag{2}$$

Notice that $\star_\rho P$ processes are special cases of $_\rho Q$ processes where Q represents a *star* process. Following the same example described in the previous section, we could use this construct to model a system having two components that may fail with probabilities varying in each time unit. Equation 3 shows a model for this system.

$$\begin{aligned} P_1 &\stackrel{def}{\equiv} {}_{\rho_1}(\mathbf{tell}\ Comp_1\_Fail) \\ P_2 &\stackrel{def}{\equiv} {}_{\rho_2}(\mathbf{tell}\ Comp_2\_Fail) \\ \mathbf{DEVICE} &\stackrel{def}{\equiv} SYSTEM||!P_1||!P_2 \end{aligned} \tag{3}$$

3.3. Stochastic parameters in the linear-temporal logic

Since we introduce uncertainty in the execution of **sntcc** processes, properties (and proofs) in the calculus must deal with probabilities. For example, given the stochastic system in section 3.1, we could verify that eventually the whole system fails because of failures in some of its components and also that failures in the first component will be more frequent. To be able to prove this kind of property we add a probability ρ to formulae. These will be tuples of the form $\langle A\ ,\ \rho\rangle$ where A is a formula in the

ntcc lineal-temporal logic. The grammar of formulae is as follows:

$$A', B', .. = \langle c, \rho\rangle \mid A' \overset{\circ}{\Rightarrow} B' \mid \overset{\circ}{\neg} A' \mid \overset{\circ}{\exists}_x A' \mid \circ A', \mid \Diamond A' \mid \Diamond A'$$

Probability ρ should not be confused with some notion of *degree of validity* of a formula. This probability refers to the occurrence of events (time). For example, the formula $\langle c < 3, 0.5\rangle$ expresses that with a probability of 0.5 constraint $c < 3$ will be asserted. It what follows, capital letters with a prime (A') will denote formulae in this new logic.
The semantics is the following:

$$\begin{array}{lll}
\langle \alpha\ ,\ i\rangle \models \langle c\ ,\ \rho\rangle & iff & \alpha(i) \models \langle c\ ,\ \rho\rangle \\
\langle \alpha\ ,\ i\rangle \models \overset{\circ}{\neg}\langle A\ ,\ \rho\rangle & iff & \langle \alpha\ ,\ i\rangle \models \langle \dot{\neg} A\ ,\ 1-\rho\rangle \\
\langle \alpha\ ,\ i\rangle \models \langle A_1\ ,\ \rho_1\rangle \overset{\circ}{\Rightarrow} \langle A_2\ ,\ \rho_2\rangle & iff & \langle \alpha\ ,\ i\rangle \models \langle A_1\ ,\ \rho_1\rangle\ implies \\
 & & \langle \alpha\ ,\ i\rangle \models \langle A_2\ ,\ \rho_2\rangle \\
\langle \alpha\ ,\ i\rangle \models \circ\langle A\ ,\ \rho\rangle & iff & \langle \alpha\ ,\ i+1\rangle \models \langle A\ ,\ \rho\rangle \\
\langle \alpha\ ,\ i\rangle \models \Box\langle A\ ,\ \rho\rangle & iff & \forall_{j\geq i}\langle \alpha\ ,\ j\rangle \models \langle A\ ,\ \rho\rangle \\
\langle \alpha\ ,\ i\rangle \models \Diamond\langle A\ ,\ \rho\rangle & iff & \exists_{j\geq i}\ s.t. \langle \alpha\ ,\ j\rangle \models \langle A\ ,\ \rho\rangle \\
\langle \alpha\ ,\ i\rangle \models \overset{\circ}{\exists}_x\langle A\ ,\ \rho\rangle & iff & there\ is\ an\ x-variant\ \alpha \\
 & & of\ \alpha\ s.t.\ \langle \alpha'\ ,\ i\rangle \models \langle A\ ,\ \rho\rangle
\end{array}$$

where $\alpha(i) \models \langle c\ ,\ \rho\rangle$ is defined as: $\langle c_1\ ,\ \rho_1\rangle \models \langle c_2\ ,\ \rho_2\rangle$ iff c_1 entails c_2 in the constraint system and $\rho_2 \leq \rho_1$.

Since formulae such as $A' \overset{\circ}{\Rightarrow} B'$ are not of the form $\langle A\ ,\ \rho\rangle$, we can express equivalences between linear-temporal implication and negation operators in **sntcc** logic and their counterpart in **ntcc** logic by using properties of probability theory:

$$\begin{array}{lll}
\overset{\circ}{\neg}\langle A\ ,\ \rho\rangle & \equiv & \langle \dot{\neg} A\ ,\ 1-\rho\rangle \\
\langle A_1\ ,\ \rho_1\rangle \overset{\circ}{\wedge} \langle A_2\ ,\ \rho_2\rangle & \equiv & \langle A_1 \dot{\wedge} A_2\ ,\ \rho_1 \times \rho_2\rangle
\end{array} \tag{4}$$

Since processes in **sntcc** can be considered probabilistic independent, the $\dot{\wedge}$ operator multiplies the probabilities. Recall that in **ntcc** , and therefore in **sntcc** , temporal conjunction and disjunction are shorthands for their equivalences w.r.t temporal implication and negations, e.g $A \dot{\wedge} B \equiv \dot{\neg}(A \dot{\Rightarrow} \dot{\neg} B)$.
By using equivalences between $\Rightarrow$ and $\wedge$ we get the following equations:

$$\begin{array}{lll}
\langle A_1\ ,\ \rho_1\rangle \overset{\circ}{\Rightarrow} \langle A_2\ ,\ \rho_2\rangle & \equiv & \overset{\circ}{\neg}(\langle A_1\ ,\ \rho_1\rangle \overset{\circ}{\wedge} \overset{\circ}{\neg}\langle A_2\ ,\ \rho_2\rangle) \\
 & \equiv & \langle A_1 \dot{\Rightarrow} A_2, \\
 & & 1-\rho_1+\rho_1\times\rho_2
\end{array} \tag{5}$$

In the same way we can verify that:

$$\begin{array}{lll}
\langle A_1\ ,\ \rho_1\rangle \overset{\circ}{\vee} \langle A_2 \rho_2 & \equiv & \langle A_1 \dot{\vee} A_2, \\
 & & \rho_1+\rho_2-\rho_1\times\rho_2
\end{array} \tag{6}$$

3.4. Inference System

We extend the inference system proposed in [5] with inference rules taking into account the new form of formulae and the probabilistic operators ${}_\rho P$ and $\star_\rho P$. The rules are as follows:

$$LTELL: \mathbf{tell}\ c \vdash \langle c\ ,\ 1.0\rangle \tag{7}$$

This says that asserting a constraint c means that it can be entailed with 100% probability.

$$LPAR: \frac{P \vdash \langle A\ ,\ \rho\rangle\ \ Q \vdash \langle B\ ,\ \rho_2\rangle}{P||Q \vdash \langle A\ ,\ \rho\rangle \overset{\circ}{\wedge} \langle B\ ,\ \rho_2\rangle} \tag{8}$$

i.e. the parallel execution of two process satisfies the conjunction of the formulae of each process.

$$LLOC: \frac{P \vdash \langle A\ ,\ \rho\rangle}{\mathbf{local}\ x\ \mathbf{in}\ P \vdash \overset{\circ}{\exists}_x\langle A\ ,\ \rho\rangle} \tag{9}$$

$$LNEXT: \frac{P \vdash \langle A\ ,\ \rho\rangle}{\mathbf{next}\ P \vdash \circ\langle A\ ,\ \rho\rangle} \tag{10}$$

$$LREP: \frac{P \vdash \langle A\ ,\ \rho\rangle}{!P \vdash \Box\langle A\ ,\ \rho\rangle} \tag{11}$$

$$LCONS: \frac{P \vdash \langle A\ ,\ \rho\rangle}{P \vdash \langle B\ ,\ \rho_2\rangle}\ if\ \langle A\ ,\ \rho\rangle \overset{\circ}{\Rightarrow} \langle B\ ,\ \rho_2\rangle \tag{12}$$

$$LSTAR: \frac{P \vdash \langle A\ ,\ \rho\rangle}{\star P \vdash \Diamond\langle A\ ,\ \rho\rangle} \tag{13}$$

If c can be entailed from the store with some probability σ, i.e $store \models \langle c\ ,\ \sigma\rangle$, the rule for *unless* processes can be depicted as follows:

$$LUNL: \frac{P \vdash \langle A\ ,\ \rho\rangle}{\mathbf{unless}\ c\ \mathbf{next}\ P \vdash \langle c\ ,\ \sigma\rangle \overset{\circ}{\vee} \circ\langle A\ ,\ \rho(1-\sigma)\rangle} \tag{14}$$

For $\sum \mathbf{when}\ c_i\ \mathbf{do}\ P_i$ process, if each c_i can be entailed with certain probability σ_i, then we have:

$$LSUM: \frac{\forall i \in I\ \ P_i \vdash \langle A_i\ ,\ \rho_i\rangle}{\begin{array}{l}\sum_{i\in I}\mathbf{when}\ c_i\ \mathbf{do}\ P_i \vdash \\ \overset{\circ}{\bigvee}_{i\in I}(\langle c_i\ ,\ \sigma_i\rangle \overset{\circ}{\wedge} \langle A_i\ ,\ \rho_i\rangle) \overset{\circ}{\vee} \overset{\circ}{\bigwedge}_{i\in I}\overset{\circ}{\neg}\langle c_i\ ,\ \sigma_i\rangle\end{array}} \tag{15}$$

If we have one-element sums, we can rewrite the previous equation conveniently as:

$$LSUM: \frac{P \vdash \langle A\ ,\ \rho\rangle}{\mathbf{when}\ c\ \mathbf{do}\ P \vdash \langle c_i\ ,\ \sigma\rangle \overset{\circ}{\Rightarrow} \langle A\ ,\ \sigma\times\rho\rangle} \tag{16}$$

The following equation:

$$PPRO : \frac{P \vdash \langle A\ ,\ \rho\rangle}{{}_{\rho_2}P \vdash \langle A\ ,\ \rho \times \rho_2\rangle} \tag{17}$$

represents the inference rule for ρP processes. Since probabilities ρ and ρ_2 are independent, the probability of both occurrences is $\rho \times \rho_2$.
Finally, the rule for the $\star_\rho P$ process is:

$$PSTAR : \frac{P \vdash \langle A\ ,\ \rho\rangle}{\star_{\rho_2}P \vdash \Diamond\langle A\ ,\ \rho \times \rho_2\rangle} \tag{18}$$

4. Examples

We describe below a model for an improved version of the system proposed in section 3.1 and we illustrate the proof of a property. The system consists in two components that may fail with probabilities ρ_1 and ρ_2 respectively, while the device is on (i.e $dev\_on = true$). If a failure occurs in some component (or in both) the system will be turned off. The model of this system in **sntcc** is as follows:

$$\begin{gathered}
P1 \stackrel{def}{\equiv} \mathbf{when}\ dev_on\ \mathbf{do}\ {}_{\rho_1}\mathbf{tell}\ (c_1_fail)\\
P2 \stackrel{def}{\equiv} \mathbf{when}\ dev_on\ \mathbf{do}\ {}_{\rho_2}\mathbf{tell}\ (c_2_fail)\\
P3 \stackrel{def}{\equiv} \mathbf{when}\ c_1_fail \vee c_2_fail\ \mathbf{do\ next\ tell}\ (dev_off)\\
P4 \stackrel{def}{\equiv} \mathbf{unless}\ c_1_fail \vee c_2_fail\ \mathbf{next\ tell}\ (dev_on)\\
\mathbf{DEVICE} \stackrel{def}{\equiv} !P1||!P2||!P3||!P4||\mathbf{tell}\ (dev_on)
\end{gathered} \tag{19}$$

Process $\mathbf{tell}\ (dev\_on)$ is added to the device definition to provide the initial stimulus. Now, we can prove that if the system is on, it may turned off in the next time unit with a certain probability ρ_x, i.e, $DEVICE \vdash \langle dev\_on\ ,\ 1.0\rangle \stackrel{\circ}{\Rightarrow} \Box \circ \langle dev\_off\ ,\ \rho_x\rangle$. To accomplish this task we use the **sntcc** logic inference system:

$$\begin{array}{c}
PPRO\dfrac{\mathbf{tell}\ c_1_fail \vdash \langle c_1_fail\ ,\ 1\rangle}{{}_{\rho_1}\mathbf{tell}\ c_1_fail \vdash \langle c_1_fail\ ,\ \rho_1\rangle}\\
LSUM\ \ P1 \vdash \langle dev_on\ ,\ 1\rangle \stackrel{\circ}{\Rightarrow} \langle c_1_fail\ ,\ \rho_1\rangle
\end{array} \tag{20}$$

In the same way we can prove that:

$$P2 \vdash \langle dev\_on\ ,\ 1\rangle \stackrel{\circ}{\Rightarrow} \langle c_2\_fail\ ,\ \rho_2\rangle \tag{21}$$

Given that the device is on, process $P1||P2$ satisfies the formula:

$$LCONS\frac{(20)\quad (21)}{P1||P2 \vdash \langle c_1\_fail\ ,\ \rho_1\rangle \stackrel{\circ}{\wedge} \langle c_2\_fail\ ,\ \rho_2\rangle} \tag{22}$$

On the other hand $P3$ satisfies the following formula:

$$\begin{array}{c}
LNEXT\dfrac{\mathbf{tell}\ dev_off \vdash \langle dev_off\ ,\ 1\rangle}{\mathbf{next\ tell}\ dev_off \vdash \circ\langle dev_off\ ,\ 1\rangle}\\
LSUM\ \ P3 \vdash \langle c_1_fail \dot{\vee} c_2_fail\ ,\ 1\rangle \stackrel{\circ}{\Rightarrow} \circ\langle dev_off\ ,\ 1\rangle
\end{array} \tag{23}$$

According to 22, we can deduce that $c1\_fail$ and $c2\_fail$ will be asserted with probabilities ρ_1 and ρ_2, respectively. Therefore, the probability of deducing formula $c_1\_fail \vee c_2\_fail$ is $\rho_1 + \rho_2 - \rho_1 \times \rho_2$. By using equations 6 and 23 and rules $LPAR$ and $LREP$ in the system definition , we can assert that:

$$\begin{gathered}
DEVICE \vdash\\
\langle dev_on\ ,\ 1\rangle \stackrel{\circ}{\Rightarrow} \Box \circ \langle dev_off\ ,\ \rho_1 + \rho_2 - \rho_1 \times \rho_2\rangle
\end{gathered} \tag{24}$$

which means that while the system is on, there is in each time unit a probability of $\rho_1 + \rho_2 - \rho_1 \times \rho_2$ that some component (or both) fails and thus that the system will be turned off.
In [6], **sntcc** is used to model a non-trivial biological system: the gene expression of the virus lambda. In this reference, readers can find a more elaborated example of **sntcc** constructs and proofs of some interesting properties that are out of the scope of this article.

5. Concluding remarks and future work

In this paper we proposed a new stochastic non-deterministic temporal concurrent constraint calculus called **sntcc** . This calculus allows us to model discrete timed concurrent systems and specify actions according to a given probability. This calculus is obtained from **ntcc** by adding two new operators and a stochastic function to the signature of the constraint system.
A new stochastic linear-temporal logic is proposed to prove properties over timed systems modeled with **sntcc** . This logic is equipped with an inference system to verify if a **sntcc** process P satisfies a formula in the logic (i.e $P \models A'$). Additionally, this inference system can be used to reason about the probability of executing P or about particular formulae satisfied by P. As an example, we showed how to model a simple electronic device with two components that may fail with different probability. Also, a proof of a property of this system was given to illustrate the use of our inference system.
We are interested in using the calculus presented here to model biological systems. We have already successfully modeled the gene expression behavior in a non-trivial biological system called the λ-switch (see [1] for a description of the system , [3] for a stochastic pi-calculus model of it and [6] for a model using **sntcc**). We plan to build a processes simulator for **sntcc** to better visualize the behavior of those system and to make quantitative measures

that can be compared with empirical results reported in the literature. Additionally we plan to use our proof system to verify properties of relevance to biologists. Adapting or implementing a new automatic theorem-prover will help us in this task.

References

[1] Adam Arkin, John Rossb, and Harley H. McAdams. Stochastic kinetic analysis of developmental pathway bifurcation in phage lambda-infected escherichia coli cells. In *Genetics*, 1998.

[2] Vineet Gupta, Radha Jagadeesan, and Vijay A. Saraswat. Probabilistic concurrent constraint programming. In *International Conference on Concurrency Theory*, pages 243–257, 1997.

[3] Celine Kuttler, Joachim Niehren, and Ralf Blossey. Gene regulation in the pi calculus: Simulating cooperativity at the lambda switch. In *Bio-CONCUR 2004*, 2004.

[4] Z. Manna and A. Penueli. *The Temporal Logic of Reactive and Concurrent Systems, Sepecification*. Springer, 1991.

[5] Mogens Nielsen, Catuscia Palamidessi, and Frank D. Valencia. Temporal concurrent constraint programming: Denotation, logic and applications. In *Special Issue of Selected Papers from EXPRESS'01, Nordic Journal of Computing*, 2001.

[6] Carlos Olarte and Camilo Rueda. Using stochastic ntcc to model biological systems. In *proceeding of CLEI 2005, Cali-Colombia*, 2005.

[7] C. Priami. Stochastic pi-calculus. In *Computer Journal*, 2004.

[8] J. Parrow R. Milner and D. Walker. A calculus of mobile processes, Parts I and II. *Journal of Information and Computation*, 100:1–77, September 1992.

[9] V. Saraswat, R. Jagadeesan, and V. Gupta. Fundation of timed concurrent constraint programming. In *IEEE Symposium on Logic in Computer Science*. IEEE press, 1994.

[10] V. Saraswat, M. Rinard, and P. Panangaden. Semantic foundations of concurrent constraint programming. In *18th Annual ACM Symposium on Principles of Programming Languages*, pages 333–353. ACM Press, 1991.

[11] V. A. Saraswat. *Concurrent Constraint Programming*. The MIT Press, Cambridge, MA, 1993.