Observable Behavior for the Verification of Probabilistic Concurrent Processes

Jorge A. Pérez and Camilo Rueda

Pontificia Universidad Javeriana, Cali – Colombia

Abstract

We propose to verify probabilistic processes by relating the observable behavior of a process (what its environment can perceive from its execution) and the notion of satisfaction of a temporal logic. We do so in a non-deterministic, timed process calculus with probabilistic choice. In addition to combine both non-deterministic and probabilistic choices, the proposed calculus allows to define a parameterizable notion of probabilistic eventual. We give an operational semantics for the calculus based on the probabilistic automaton model. Such a semantics allows to interpret the observable behavior of a process as the structures upon which satisfaction is defined in Hansson and Jonsson’s PCTL.

1 Introduction

The description of probabilistic behavior has shown to be essential in emerging application areas such as, e.g., systems biology and computer music, where is common to find large amounts of quantitative information associated to behavior. Not surprisingly, the inclusion of stochastic or probabilistic parameters within a number of formalisms and languages has been studied (see, e.g., [9,8,6,3]). What most of such extensions do is to associate some priority or rate to processes execution. By only such an association, however, it is difficult to analyze those systems where behavior (i) tends to follow non-trivial behavior patterns, such as those induced by or involving non-determinism and/or asynchrony, and (ii) depends on partially known and/or empirical information. This is the case of, for instance, genetic regulatory networks [1] and music improvisation systems [10]. There is then the need of studying probabilistic behavior in the context of models for concurrency with richer representations of quantitative information.

We are interested in the study of models for concurrency considering expressive, flexible forms of quantitative information in their languages and associated techniques. One model for concurrency that is particularly well-suited for the study of systems involving partial information is concurrent constraint programming (ccp) [11]. In ccp systems are naturally described by constraints, pieces of partial information that can involve explicit quantitative parameters (such as, e.g., $x \leq 43$). Concurrent agents interact by adding new constraints and by synchronizing on the partial information.

©2006 Published by Elsevier Science B. V.
they represent. Quantitative parameters are supported in ccp by means of constraint systems, parameterizable theories defining interdependencies between them. All this suggests that complementing ccp-based calculi with probabilistic behavior is reasonable: it could effectively enhance the accuracy of specifications and the significance of derived proofs.

A prominent feature of ccp is that processes enjoy a dual nature: they can be seen at the same time as computing agents and logic formulas. As such, any syntactical and semantic complements to the model should be accompanied by a logic counterpart. In our case, it is important that such a counterpart exploits the probabilistic information in models as much as possible. The derived logical properties should be amenable for automated verification. Finding ways of associating model checking techniques to probabilistic ccp-based calculi is thus a must. In this context, the conceptual similarities between some variants of timed ccp calculi and some probabilistic, temporal logics equipped with model-checking techniques suggest their combination.

In this work we explore such similarities to propose a framework that allows to describe timed systems exhibiting both non-deterministic and probabilistic behavior and, in the ccp spirit, also allows to reason about them in terms of properties of processes that explicitly involve time bounds and probabilities.

To do so we provide a simple (yet formal) association between a timed ccp calculus and a probabilistic temporal logic. The calculus (called prob-ntcc) is a probabilistic extension of ntcc [7], a formalism capable of representing non-deterministic behavior in a timed context. The extension is thus capable of representing both non-deterministic and probabilistic behavior and allows to define a parametrizable notion of probabilistic eventuality. Such a notion represents the eventual execution of processes based on probabilistic choices and recursion, with the possibility of direct user control. prob-ntcc is complemented by Hansson and Jonsson’s PCTL [5], a logic that reasons in terms of soft deadlines —statements explicitly involving a time bound and a probability. We believe that from the interaction of probabilistic and non-deterministic choices, together with the soft deadlines approach for verification, we obtain an interesting framework for the analysis of concurrent systems involving explicit quantitative information.

Contributions The overall contribution of this work is the formal association between a probabilistic, timed process calculi (prob-ntcc) and PCTL, a logic capable of reasoning about both discrete time and probabilities and that includes efficient model checking procedures. Central to this contribution is the definition of prob-ntcc and its operational semantics. Such a semantics, based on the probabilistic automaton model [12], makes it possible to interpret the observable behavior of a process as a discrete-time Markov Chain (DTMC). Since satisfaction in PCTL is defined in terms of DTMCs, it becomes possible to reason about prob-ntcc processes in terms of expressive PCTL soft deadlines.

Structure of the document Some preliminary background that will be used later on the paper is given next. Section 3 describes the syntax and semantics of prob-ntcc. The formal interpretation of the observable behavior as a DTMC is given there. Section 4 illustrates the use of prob-ntcc with a simple example. Section 5 concludes.
2 Preliminaries

We introduce some background that will be used later on the paper: concepts from probability theory, a description of the Probabilistic Automaton (PA) model [12] as well as an introduction to the PCTL logic [5].

2.1 Probability Theory

Given a set $C$, a field on $C$ (denoted by $F$), is a family of subsets of $C$ that contains $C$, and that is closed under complementation and finite union. A $\sigma$-field (or $\sigma$-algebra) on $C$ (denoted by $F$), is a field on $C$ that is closed under countable union. The elements of a $\sigma$-field are called measurable sets. The pair $(C, F)$ is called a measurable space.

Let $C$ be a family of subsets of $\Omega$. A measure $\mu$ on $C$ is a function that assigns a non-negative real-value (possibly $\infty$) to each element of $C$, such that 1) if $\emptyset$ is an element of $C$, then $\mu(\emptyset) = 0$ and 2) if $(C_i)_{i \in \mathbb{N}}$ forms a sequence of pairwise disjoint elements of $C$, and $\cup_i C_i$ is an element of $C$, then $\mu(\cup_i C_i) = \sum_i \mu(C_i)$. If $(\Omega, F)$ is a measurable space, then a measure on $F$ is called a measure on $(\Omega, F)$.

A probability space is a triple $(\Omega, F, P)$, where $(\Omega, F)$ is a measurable space, and $P$ is a measure on $(\Omega, F)$ such that $P(\Omega) = 1$. The measure $P$ is also referred to as a probability measure or a probability distribution. The set $\Omega$ is called the sample space, and the elements of $F$ are called events. The measure of an event $E$ is denoted by $P[E]$. For the sake of clarity, $P_*$ will denote the probabilistic space defined by $(\Omega_*, F_*, P_*)$.

2.2 Probabilistic Automata

Probabilistic automata have been proposed in [12]. This model provides a unified framework for handling non-deterministic and probabilistic choices. Formally, a probabilistic automaton $M$ is defined as follows:

(i) A set $\text{states}(M)$ of states,
(ii) A non-empty set $\text{start}(M) \subseteq \text{states}(M)$ of start states,
(iii) An action signature $\text{sig}(M) = (\text{ext}(M), \text{int}(M))$, where $\text{ext}(M)$ and $\text{int}(M)$ are disjoint sets of internal and external actions, respectively. With these sets, the set of actions of $M$ is defined as $\text{acts}(M) = \text{int}(M) \cup \text{ext}(M)$,
(iv) A transition relation $\text{trans}(M) \subseteq \text{states}(M) \times \text{Probs}((\text{acts}(M) \times \text{states}(M)) \cup \{\delta\})$.

For a set $C$, $\text{Probs}(C)$ denotes the set of discrete probabilistic spaces $(\Omega, F, P)$ with no 0-probability elements such that $\Omega \subseteq C$. The elements of $\text{trans}(M)$ are called transition groups or steps.

A probabilistic automaton differs from an ordinary one only in the transition relation. For each state $s$, once a transition group has been chosen non-deterministically, the action that is performed and the state that is reached are determined probabilistically, by means of a discrete probability distribution. This is how both kinds of choices coexist in the same model. Each transition group $(s, \mathcal{P})$ may contain a special symbol $\delta$, which represents the possibility for the system not to complete the transition.

Probabilistic automata can be either simple or fully probabilistic. In a simple probabilistic automaton, for each transition group $(s, \mathcal{P})$ of $\text{trans}(M)$ there is an action of $M$ such that $\Omega \subseteq \{a\} \times \text{states}(M)$, i.e., each transition is associated with a single action.
and it always completes. Once a transition group is chosen, then only the next state is chosen probabilistically. A probabilistic automaton $M$ is fully probabilistic if $M$ has a unique start state, and from each state of $M$ there is at most one transition group enabled. A fully probabilistic automaton does not contain any non-determinism.

One concept that is central to simple probabilistic automata is that of a scheduler or adversary. Intuitively, the task of a scheduler is to resolve the non-determinism in the automaton, based on the history of its execution. More precisely, in a (simple) probabilistic automaton $M$ in the state $s$, a scheduler $Sch$ will select a specific transition group $(s, P_i)$ from $\{s, \text{Prob}(\text{acts}(M) \times \text{states}(M))\}$, the set of all transition groups starting in $s$. There are no restrictions on the nature of this selection; it is usually left unspecified to model any scheduling policy.

### 2.3 PCTL: A temporal logic for reasoning about soft deadlines

We now summarize the logic PCTL, based in [5]. Very broadly, PCTL allows to reason about properties such as “after a request for a task, there is at least a 95 percent that the task will be accomplished within 5 minutes”. This kind of statements, explicitly defining both a probability for a process and a time bound are known as soft deadlines. Unlike hard deadlines, soft deadlines are meant to characterize systems in which a failure does not imply catastrophic consequences such as loss of lives or economical disasters.

#### Syntax and Semantics

Formulas in PCTL are built from a set $A$ of atomic propositions, propositional logic connectives and operators for expressing time and probabilities. They are divided into state and path formulas. While the former represent properties of states, the latter represent properties of sequences of states. Their syntax can be given as follows:

- Each atomic proposition (denoted by $a_i$) is a state formula
- For two state formulas $f_1$ and $f_2$, then $\neg f_1$, $(f_1 \land f_2)$, $(f_1 \lor f_2)$, $(f_1 \rightarrow f_2)$ are also state formulas
- For two state formulas $f_1$ and $f_2$ and a non-negative number $t$ (or $\infty$), $(f_1 \U^{\leq t} f_2)$ and $(f_1 \U^{> t} f_2)$ are path formulas
- Given a path formula $f$ and a real number $p$ (such that $0 \leq p \leq 1$), then $[f]_{\geq p}$ and $[f]_{> p}$ are state formulas

The propositional operators have the usual meaning. $U$ and $\U$ are the strong and weak until operators, respectively. For a state $s$, the formula $[f]_{\geq p}$ ($(f]_{> p}$) expresses that $f$ holds for a path starting in $s$ with a probability of at least (greater than) $p$.

It is customary to abbreviate formulas $[f_1 \U^{\leq t} f_2]_{\geq p}$ and $[f_1 \U^{\leq t} f_2]_{> p}$ with $f_1 \U^{\leq t}_{\geq p} f_2$ and $f_1 \U^{> t}_{> p} f_2$, respectively. The formula $[f_1 \U^{\leq t}_{> p} f_2]_{\geq p}$ means that there is at least a probability $p$ that both $f_2$ will become true within $t$ time units and $f_1$ will be true from now on until $f_2$ becomes true. On the other hand, formula $[f_1 \U^{\leq t}_{> p} f_2]_{> p}$ means that there is at least a probability $p$ that either $f_1$ will remain true for at least $t$ time units, or both $f_2$ will become true within $t$ time units and $f_1$ will be true from now on until $f_2$ becomes true. Formulas $f_1 \U^{t}_{\geq p} f_2$ and $f_1 \U^{t}_{> p} f_2$ have an analogous meaning.

PCTL formulas are interpreted over structures that are discrete-time Markov chains.
Definition 1 (Structures) A structure in PCTL is a quadruple \( \langle S, s^i, \mathcal{T}, L \rangle \), where

- \( S \) is a finite set of states and \( s^i \in S \) is the initial state,
- \( \mathcal{T} \) is a transition function, \( \mathcal{T} : S \times S \rightarrow [0,1] \), \( s, t \) for all \( s \in S \), \( \sum_{t \in S} \mathcal{T}(s, s') = 1 \),
- \( L \) is a labeling function assigning atomic propositions to states: \( L : S \rightarrow 2^1 \).

In a structure each one of the transitions is considered to require one time unit. It is usual to represent structures as diagrams where states are depicted as nodes and transitions are represented by directed arcs labeled with their associated probabilities.

We now give some probability-related concepts over structures. A path \( \sigma \) from a state \( s_0 \) in a structure is an infinite sequence \( s_0, s_1, \ldots, s_n, \ldots \) of states such that \( s_0 \) is the first state. The \( n \)-th state of \( \sigma \) \((s_n)\) is denoted by \( \sigma[n] \), while the prefix of \( \sigma \) of length \( n \) is denoted \( \sigma \uparrow n \). For each structure and state \( s_0 \), a probability measure \( \mu_m \) on the set of paths from \( s_0 \) is defined. \( \mu_M \) is defined on the measurable space \( \langle X, \mathcal{A} \rangle \), where \( X \) is the set of paths starting in \( s_0 \) and \( \mathcal{A} \) is a \( \sigma \)-field on \( X \) generated by sets \( \{ \sigma \in X : \sigma \uparrow n = s_0, s_1, \ldots, s_n \} \) of paths with a common finite prefix \( s_0, s_1, \ldots, s_n \). The measure \( \mu_m \) is defined as follows: for each finite sequence \( s_0, s_1, \ldots, s_n \) of states,

\[
\mu_m(\{\sigma \in X : \sigma \uparrow n = s_0, s_1, \ldots, s_n\}) = \mathcal{T}(s_0, s_1) \times \cdots \times \mathcal{T}(s_{n-1}, s_n).
\]

For \( n = 0 \), \( \mu_m(\{\sigma \in X : \sigma \uparrow 0 = s_0\}) = 1 \). This uniquely defines \( \mu_m \) on all sets of paths in \( \mathcal{A} \).

Given a structure \( K \), truth for formulas is defined by a satisfaction relation \( s \models_K f \). A satisfaction relation for states, denoted \( \sigma \models_K f \), is also defined (\( \sigma \) and \( f \) being a path and a path formula, respectively). These two relations are defined in Table 1. Finally, it is defined that \( s \models_K f \equiv s^i \models_K f \), where \( s^i \) is the initial state of \( K \).

Properties Expressible in PCTL

In PCTL it is possible to state that a property will hold continuously during a specific time interval, or that a property will hold sometime during a time interval:

\[
G_{\geq p}^{\leq t} f \equiv f U_{\geq p}^{\leq t} \text{false} \quad F_{\geq p}^{\leq t} f \equiv \text{true } U_{\geq p}^{\leq t} f.
\]

Informally, \( G_{\geq p}^{\leq t} f \) means that \( f \) holds continuously for \( t \) time units with a probability of at least \( p \), and \( F_{\geq p}^{\leq t} f \) means that \( f \) hold within \( t \) time units with a probability of at least \( p \). Another requirement on most reactive systems is that they should be continuously operating. The following formulas express such properties:
Pêrez, Rueda

\[
AG \ f \equiv f \cup_{\leq \infty} false \\
AF \ f \equiv true \cup_{\geq 1} f \\
EG \ f \equiv f \cup_{\geq 0} false \\
EF \ f \equiv f \cup_{\geq 0} true
\]

Intuitively, \(AG\ f\) means that \(f\) is always \textit{true}, \(AF\ f\) means that a state where \(f\) is \textit{true} will eventually be reached with a probability 1, \(EG\ f\) means that there is a non-zero probability for \(f\) to be continuously \textit{true} and \(EF\ f\) means that there exists a state where \(f\) holds which can be reached with a non-zero probability. To describe reactive systems the \textit{leads-to} operator may come in handy. It represents the fact that whenever a formula holds, another one will eventually hold. This is formalized in PCTL as follows:

\[
f_1 \leadsto_{\geq t} f_2 \equiv AG [(f_1 \rightarrow F^{\leq t}_{\geq p} f_2)].
\]

Intuitively, whenever \(f_1\) holds there is a probability of at least \(p\) that \(f_2\) will hold within \(t\) time units.

3 A probabilistic, timed ccp process calculus

Here we shall describe the syntax and operational semantics for \texttt{prob–ntcc}, a probabilistic extension of \texttt{ntcc} \cite{7}. We start by introducing some general concepts for ccp-based calculi.

A \textit{constraint system} is a theory specifying interdependencies between pieces of information (constraints); e.g., from \(x > 27\) we can infer \(x > 0\). More formally, a constraint system (cs) is a pair \((\Sigma, \Delta)\) where \(\Sigma\) is a signature of function and predicate symbols, and \(\Delta\) is a decidable theory over \(\Sigma\) (i.e., a decidable set of sentences over \(\Sigma\) with at least one model). Given a cs \((\Sigma, \Delta)\), let \((\Sigma, V, S)\) be its underlying first-order language, where \(V\) is a set of variables \(x, y, \ldots\), and \(S\) is the set of logic symbols \(\neg, \wedge, \lor, \Rightarrow, \exists, \forall, true\) and \(false\). Constraints \(c, d, \ldots\) are formulas over this first-order language. We say that \(c\) \textit{entails} \(d\) in \(\Delta\), written \(c |\Rightarrow d\), iff \(c \Rightarrow d\) is true in all models of \(\Delta\). The relation \(\models\), which is decidable by the definition of \(\Delta\), induces an equivalence \(\approx\) given by \(c \approx d\) iff \(c \models d\) and \(d \models c\). Henceforth, \(C\) denotes the set of constraints under consideration modulo \(\approx\) in the underlying cs. Thus, we simply write \(c = d\) iff \(c \approx d\).

In \texttt{ntcc} and \texttt{prob–ntcc} time is divided into \textit{discrete intervals (or time units)}. In a particular timed interval, a process \(P\) gets an input (an item of information represented as a constraint) \(c\) from the environment, it executes with this input as the initial \textit{store}, and when it reaches its resting point, it \textit{outputs} the resulting store \(d\) to the environment. The resting point determines a residual process \(Q\), which is then executed in the next time interval. Information is not automatically transferred from one time unit to another.

3.1 Syntax and Informal Meaning of Processes

Processes \(P, Q, \ldots \in \text{Proc}\) are built from constraints \(c \in C\) and variables \(x \in V\) in the underlying constraint system by the following syntax.

\[
\begin{align*}
P, Q, \ldots := & \text{tell}(c) \mid \sum_{i \in I} \text{when } c_i \text{ do } (P_i, a_i) \mid P \parallel Q \mid !P \\
& \mid \text{local } x \text{ in } P \mid \text{next } (P) \mid \text{unless } c \text{ next } (P)
\end{align*}
\]
Process `tell(c)` adds the constraint `c` to the current store, thus making `c` available to other processes in the current time interval. Process \( \sum_{i \in I} \text{when } a_i \text{ do } (P_i, a_i) \), where `I` is a finite set of indexes and every `a_i \in [0, 1]` is a real number, represents a probabilistic choice. In the current time interval, this process probabilistically chooses one of the `P_j` \((j \in I)\) according to the distribution defined by the `a_i`'s. Each `a_i` thus represents the probability of choosing `P_i` for execution. The guards that can be entailed from the current store determine the enabled processes that are considered in such a distribution. The chosen alternative, if any, precludes the others. If no choice is possible then the summation is precluded. We use `skip` as an abbreviation of the empty summation and "+" for binary summations.

Process `P || Q` represents the parallel composition of `P` and `Q`. In one time interval `P` and `Q` operate concurrently, “communicating” via the common store. We use `\prod_{i \in I} P_i`, where `I` is finite, to denote the parallel composition of all `P_i`. Process `local x 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`.

Process `next (P)` represents the activation of `P` in the next time interval, i.e., a one-unit delay. The process `unless c next (P)` is similar, but `P` will be activated only if `c` cannot be inferred from the current store. The “unless” processes wait one time unit for a piece of information `c` to be present and if it is not, they trigger activity in the next time interval. We use `next^n (P)` as an abbreviation for `next (next (. . . next (P)) . . . )`, where `next` is repeated `n` times. Infinite behavior through the time intervals is represented by means of the `replication` operator `!: P`. `!P` represents `P || next (P) || next^2P || . . .`, i.e. unboundedly many copies of `P` but one at a time.

**Useful Encodings**
The following encodings proposed for `ntcc` can also used in `prob—ntcc`.

**Recursion** It is possible to encode recursive definitions of the form `q(x) \text{ 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”. Moreover, we can rely on the usual intuitions concerning procedure calls in a programming language.

**Cells** A cell can be thought of as a 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. Notations `x : v` and `x := v` represent the initialization and the assignment of a cell `x` with value `v`, respectively. Also, we use notation `x = g(x)` to abbreviate the assignment `x := g(x')`, where `x'` is the value of the cell `x` in the previous time unit and `g` is a function.

**A Probabilistic Eventuality Operator.**
In `ntcc` the `*` operator represents eventuality (or unbounded asynchrony): process `* P` represents the eventual (but certain) execution of `P` in an undetermined future time unit. As such, it constitutes another source of non-determinism in the calculus. To cope with this kind of behavior in `prob—ntcc`, we define a probabilistic eventuality operator, encoded by means of probabilistic choices:
\[ *_{(f,r)} P \overset{\text{def}}{=} \text{when } r < 1 \text{ do} \]
\[ (\text{when true do } (P,r) + \text{when true do } (\text{next } *_{(f,r)} P, 1 - r), 0.5) \]
\[ + \text{when } r \geq 1 \text{ do } (P, 0.5) \]

This derived operator relies on recursion to express controlled delays. It depends on two parameters, \( f \) and \( r \). \( r \) stands for the probability of executing \( P \): the closest to 1 \( r \) is, the greater the probability of executing \( P \) will be. Conversely, \( 1 - r \) denotes the probability of delaying \( P \)'s execution. \( f \) is a fixed function which governs the execution of \( P \) by modifying \( r \) in each recursive call. The outermost binary choice depends on two mutually exclusive guards (\( r < 1, r \geq 1 \)). This ensures that the enabled process will be executed with probability 1 (because of the normalization of the probabilities, discussed later). This also ensures finite recursive calls: if \( r \) is equal or greater than 1 then \( P \) is immediately executed. In this way, the selection of appropriate parameters for \( f \) and \( r \) might lead to complex patterns of behaviour based on (finite) recursions.

**Structural Congruence.**
As in \( \text{ntcc} \), the structural congruence relation (denoted by \( \equiv \)) for \( \text{prob–ntcc} \) is defined by the following axioms: i) \( P \parallel \text{skip} \equiv P \), ii) \( P \parallel Q \equiv Q \parallel P \), iii) \( P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R \).

### 3.2 Operational Semantics

Here we describe an operational semantics for \( \text{prob–ntcc} \). It considers process-store configurations \( \gamma \) of the form \( \langle P, c \rangle \); the set of all configurations is denoted as \( \Gamma \). Following the design rationale in the operational semantics for \( \text{ntcc} \), two transition relations are defined for \( \text{prob–ntcc} \): one internal that is meant to be hidden to the environment and an observable one that serves as an “interface” between the process and its environment. The purpose of internal transitions is to describe activity within a time unit, considering both non-deterministic and probabilistic behavior by means of a probabilistic automaton. The scheduler that has influence on such an “internal” automaton solves non-deterministic choices in such a way that the observable transition relation only reflects the probabilistic choices the system executes over time.

**Internal Transitions.** Following [6], we define the internal behavior of a \( \text{prob–ntcc} \) process \( P \) as a probabilistic automaton \( M \) whose states are the configurations \( \gamma_i \in \Gamma \) reachable from an initial configuration \( \langle P, c \rangle \) (where \( c \) is supposed to be an initial stimulus for the system represented by \( P \)). We do not consider the notion of action in the automaton (often used as a synchronization means), so transitions are unlabeled. We borrow the following notation from [6] to represent transition groups in a probabilistic automaton:

\[ \gamma_i \{ \longrightarrow_{P_i} \gamma_{i+1} \}, \]

where \( (\gamma_i, P_i) \in \text{trans} \) and \( P_i(\gamma_{i+1}) = p_i \). A full description for internal transitions is given in Table 2 (upper part). Rules PTELL, PUNL and PREP formalize the informal descriptions for tell, unless and the replication processes given before; all of them have associated probability equal to 1. A similar criteria applies for PLOC and PSTR (which represent locality and structural congruence in reductions, respectively), except for the
The fact that probability $p_i$ is taken from a previous transition. Rule PSUM formalizes probabilistic choices. The probability associated with such a choice is defined as

$$a'_j = \frac{a_j}{\sum_{k \in \{j \mid d=c_j \}} a_k},$$

where $d$ is the current store and $a_j$ is the probability of the enables process chosen, which is normalized taking into account the probabilities associated to the enabled processes. Rule PPAR formalizes parallel composition, the only source of non-determinism in the calculus.

**Observable Transition.** Before describing the observable transition rule, we shall give some definitions regarding internal transitions and configurations. The first one has to do with the transition groups enabled from a state in the probabilistic automaton.

**Definition 2 (Transition Groups from a state)** Let $M$ be a probabilistic automaton $M$. For a state $s \in \text{states}(M)$, the set of transition groups from $s$ is defined as $\text{tgroups}(s) = \{(s_i, \mathcal{P}_i) \in \text{trans}(M) \mid s = s_i\}$.

$tgroups(s)$ then represents the possible courses of execution that an external adversary can select from to implement a given scheduling policy. This idea is included in the following definition that formalizes the notion of a sequence of internal transitions.

**Definition 3 (Internal Sequence)** Let $\text{Sch} : \text{trans}(M) \rightarrow \text{trans}(M)$ be a function that given a set of transition groups, selects one of them following some scheduling policy. We define $\gamma_1 \{\rightarrow b \gamma_n\}_{\text{Sch}}$ to represent the internal sequence

$$\gamma_1 \rightarrow a_1 \gamma_2; \gamma_2 \rightarrow a_2 \gamma_3; \cdots; \gamma_{n-2} \rightarrow a_{n-2} \gamma_{n-1}; \gamma_{n-1} \rightarrow a_{n-1} \gamma_n$$

where $b = a_1 \cdot \cdots \cdot a_{n-1}$ and for every $\gamma_{i+1}$ there exists $\mathcal{P}_{i+1}$ such that $\text{Sch}(\text{tgroups}(\gamma_i)) = (\gamma_{i+1}, \mathcal{P}_{i+1})$.

An internal sequence thus describes a particular result of the probabilistic choices under the influence of a scheduler within a time unit.

Rule POBS in Table 2 (lower part) assumes a particular internal sequence. Such a sequence must not admit further computations (which is denoted by $\not \rightarrow$). The rule, for an internal sequence starting in $\langle P, c \rangle$ and finishing in $\langle Q, d \rangle$, defines an (observable) evolution $P \xrightarrow{\langle c,d,a \rangle} \text{Sch} R$. In the description of an observable evolution the scheduler used to solve the non-deterministic choices is explicitly defined. This makes sense if one considers that using a different scheduler could lead to a different final configuration. Sometimes we write $P \xrightarrow{\langle c,d,a \rangle} \text{Sch} R$ instead of $P \xrightarrow{\langle c,d,a \rangle} R$ when $\text{Sch}$ is unimportant or can be inferred from the context; the presence of a scheduler in internal computations, however, must be always assumed. Also notice that, even considering the same scheduler, $\langle Q, d \rangle$ above is but one of the possible configurations that might result from probabilistic choices. Finally, and as in ncc, the process to be executed in the next time interval is $F(Q)$, the “future” of process $Q$. $F(Q)$ is obtained by removing from $Q$ summations that did not trigger activity and any local information which has been stored in $Q$, and by “unfolding” the sub-terms within “next” and “unless” expressions.
### Table 2

Internal and observable transition rules for prob-ntcc.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PTELL</strong></td>
<td>$(\text{tell}(d),c) \rightarrow_1 \langle \text{skip},c \land d \rangle$</td>
</tr>
<tr>
<td><strong>PPAR</strong></td>
<td>$(P,c) \rightarrow_1 \langle P',c' \rangle$</td>
</tr>
<tr>
<td><strong>PREP</strong></td>
<td>$(P;e) \rightarrow_1 \langle P \parallel Q, e \rangle$</td>
</tr>
<tr>
<td><strong>PSTR</strong></td>
<td>$\frac{\gamma_1 \rightarrow_1 \gamma_0}{\gamma_1 \rightarrow_1 \gamma_2}$ if $\gamma_1 \equiv \gamma_1'$ and $\gamma_2 \equiv \gamma_2'$</td>
</tr>
<tr>
<td><strong>PSUM</strong></td>
<td>$\sum_{i \in I} \text{when } c_i \text{ do } (P_i,c_i,d) \rightarrow_1 \langle P_j,d \rangle$ if $d \models c_j, j \in P$</td>
</tr>
<tr>
<td><strong>PUNL</strong></td>
<td>$\text{unless } c \text{ next } (P,d) \rightarrow_1 \langle \text{skip},d \rangle$ if $d \models c$</td>
</tr>
<tr>
<td><strong>PLOC</strong></td>
<td>$(P,c \land \exists d) \rightarrow_1 \langle P',c' \rangle$</td>
</tr>
<tr>
<td><strong>POBS</strong></td>
<td>$\frac{(P,c) \rightarrow_1 \langle Q,d \rangle}{\text{Sch } \overrightarrow{R} \rightarrow_1 (P,c,d)}$ if $R \models F(Q)$</td>
</tr>
</tbody>
</table>

3.3 **Observable Behavior as a Discrete-Time Markov Chain**

Here we describe a formal relationship between prob-ntcc processes and PCTL. More precisely, we show how the observable behavior of a process corresponds to a fully-probabilistic automaton. Such an automaton, in turn, can be easily interpreted as the discrete-time Markov chain (DTMC) upon which satisfaction in PCTL is defined (Definition 1). In that way, we obtain intuitive and powerful verification capabilities (in terms of soft deadlines) over prob-ntcc processes. We start by defining the possible evolutions from a given process, i.e., its alternatives.

**Definition 4 (Alternatives)** Given a process $P$, its set of alternatives is defined by $\text{Alt}(P) = \{ Q \mid P \xrightarrow{(c,d,p)} Q \}$, for any $c$, $d$ and $p$.

The following definition represents a particular execution of the system over time.

**Definition 5 (Observable Sequence)** An observable sequence $\overline{s}$ with initial configuration $(P_0,c_0)$ is a sequence of evolutions

$$P_0 \xrightarrow{(c_0,d_1,p_1)} P_1 \xrightarrow{(c_1,d_2,p_2)} \cdots P_{n-1} \xrightarrow{(c_{n-1},d_n,p_n)} P_n \cdots$$

such that every $P_{i+1} \in \text{Alt}(P_i)$.

Some useful notation over observable sequences follows. The set of processes occurring in a sequence $\overline{s}$ is denoted as $\text{Procs}(\overline{s})$. Its initial process is represented by $\text{init}(\overline{s})$. For a process $P_i \in \text{Procs}(\overline{s})$, store($P_i$) denotes its associated store $c_i$. Furthermore, let $\text{sequences}(\langle P_0,c_0 \rangle)$ denote the set of all observable sequences with initial configuration $\langle P_0,c_0 \rangle$.

As mentioned before, the observable evolutions originating in a given process do not involve non-deterministic behavior, as it has been resolved by internal evolutions. For a process $P$, its observable behavior therefore can be related to a fully-probabilistic probabilistic automaton in which states correspond to the possible evolutions of $P$, actions can be mapped to the stimuli the environment provides and the transitions of the automaton can be obtained from the alternatives of each intermediate process.
In this context, an observable sequence would correspond to a particular trace of the automaton. More formally:

**Definition 6 (Observable Automaton)** Given a process $P$ and an initial stimulus $c_0$, the behavior originating in $(P, c_0)$ is a fully-probabilistic automaton $O$ where:

- $\text{states}(O) = \bigcup_{\bar{z}_i \in \text{sequences}(\langle P, c_0 \rangle)} ^{} \text{Proc}(\bar{z}_i)$
- $P$ is the initial state
- the action signature is given by $\text{acts}(O) = \text{Constraints}$,
- there is a transition relation given by $\text{trans}(O) = \text{states}(O) \times \text{Prob}(\text{acts}(O), \text{states}(O))$.

For each state $P_i$ there is only one transition group, made of the processes in $\text{Alt}(P_i)$ and the probabilities, initial and final stores given by their respective evolutions.

Summing up, probabilistic behavior for a prob$\text{-ntcc}$ process arises at two levels, one internal that takes place within a time unit and considers non-deterministic choices, and another one that is observable to any external entity and that only reflects probabilistic choices through the time intervals. Each level is formalized by a probabilistic automaton. At this point it is easy to see how the observable automaton for a process $P$ corresponds to the DTMC that defines satisfaction in PCTL. In fact, a DTMC for a prob$\text{-ntcc}$ process corresponds to an observable automaton $O$ enriched with a labeling function $L$, such that for each $s \in \text{states}(O)$ $L(s) = \text{store}(s)$. In this way, a formal relationship between prob$\text{-ntcc}$ and PCTL is completed.

### 4 An Example

We now illustrate the use of prob$\text{-ntcc}$ in the modeling of a simple biological system and in the verification of a simple soft deadline.

As mentioned in the introduction, many phenomena in systems biology exhibit quantitative information related to behavior; as discussed in [4], timed ccp constitutes a valid alternative for describing and reasoning about them. We give a simplified model of the control system of a biological network; it governs the number of molecules interacting with the control region of a set of genes. The system can be either in wild (i.e., normal) or mutated states; in each of them, the number of molecules on the control region varies differently. We will assume that the moment in which the mutation takes place is undetermined, but the possibility of a mutation to occur tends to increase as time progresses. This description can be formalized in prob$\text{-ntcc}$ as follows:

$$
M \overset{\text{def}}{=} (\text{tell}(\text{mut} = 1) \parallel \text{next} (\text{tell}(x = f_W(x))))
$$

$$
W \overset{\text{def}}{=} \text{unless} \quad \text{mut} = 1 \quad \text{next} (\text{tell}(x = f_W(x)));
$$

$$
\text{C Region} \overset{\text{def}}{=} ^*(y, 0.7) M \parallel W \parallel \text{tell}(x = \text{init})
$$

where $g(y) = y + 0.08$ for all $0 \leq y < 1$. Variable $x$ represents the number of molecules; constant init represents its value at the start of the simulation. $f_W$ and $f_H$ are functions determining the value of $x$ in the wild and mutated states, respectively. Process $\text{C Region}$ represents the whole system: it consists of one process that initializes $x$ and two processes representing each possible state. The occurrence of a mutation is represented by the guard $\text{mut} = 1$ (posted by process $M$): it determines which one of the processes modifies $x$. Process $W$ controls $x$ as long as the guard $\text{mut} = 1$ can not be
The complete observable sequences for this process are depicted in Figure 1. At the end, there is a process that leads to the mutation state with a probability of 1.

0.86, and 0.94) or to further delays caused by new probabilistic eventualities processes. The system might evolve either to a mutation state (with increasing probabilities of 0.78, 0.86, and 0.94) or to further delays caused by new probabilistic eventualities processes. At the end, there is a process that leads to the mutation state with a probability of 1. The complete observable sequences for this process are depicted in Figure 1.

We now illustrate the verification of a soft deadline of the CRegion process, based on the observable behavior analyzed above. We start by extracting the components of the DTMC. From the five observable sequences already described we can extract the states of the structure. For the sake of readability, we denote the set states(O) = \{s_0, \ldots, s_5\} as follows:

\[
\begin{align*}
    s_0 & = CRegion \\
    s_1 & = *_{(g, 0.78)} M \\
    s_2 & = M || W || tell(x = f_0(x)) \\
    s_3 & = *_{(g, 0.86)} M || W || tell(x = f_0(x)) \\
    s_4 & = *_{(g, 0.94)} M || W || tell(x = f_0(x)) \\
    s_5 & = *_{(g, 1.2)} M || W || tell(x = f_0(x))
\end{align*}
\]

with s_0 as the initial state. Both the action signature and the transition relation can be easily identified from Figure 1; the DTMC expressing the behavior of the CRegion
process is depicted in Figure 2 below.

Fig. 2. Discrete-time Markov chain derivable from process CRegion.

The above figure opens the way for verification of expressive soft deadlines. We are interested in one relating the occurrence of a mutation with a fixed probability value, namely

given the \texttt{prob-ntcc} process \textit{CRegion} there will be a mutation within three units with a probability of at least 0.9.

Given the above DTMC, this property is expressed in PCTL as:

\[ f = \text{true} U_{\geq 0.9} s_2. \]

In [5] efficient algorithms are proposed to determine whether a PCTL formula holds. Broadly speaking, such algorithms focus on labeling the states of the DTMC of the system in question with the sub-formulas that are true in each state. The formula is said to hold if the initial state is labeled with the desired formula. Following one of such algorithms, it is easy to check that \( f \) holds for our \textit{CRegion} process.

5 Concluding Remarks

We have described how a probabilistic timed process calculus and a probabilistic logic can be related by means of the observable behavior of processes. We have proposed \texttt{prob-ntcc}, a probabilistic CCP-based process calculus that combines non-deterministic and probabilistic choices in a timed context. The inclusion of a probabilistic choice operator enhances the capabilities of timed CCP for describing and reasoning about systems involving quantitative information. Indeed, since often probabilistic behavior serves to give coherence to empirical/incomplete information, including it in a framework which is by itself a convenient alternative for analyzing systems based on partial information is a natural direction. The calculus is capable to express eventuality in terms of probabilistic choices and recursion; complex patterns of behavior can be succinctly described.
by means of the parameters governing such choices. We gave a simple biological example where both this probabilistic eventuality and some basic verification capabilities are demonstrated.

The relationship between prob-ntcc with PCTL paves the way for an alternative approach for verification of reactive systems. Our immediate plans involve implementing software tools that allow to simulate/verify prob-ntcc models in practice, exploiting the model-checking procedures associated to PCTL as well as an existing tool for simulating and programming timed ccp processes [2]. The current state of the tool makes it straightforward to build the DTMC required for verification in PCTL. We also plan to explore real applications in areas such as computer music so to test both our approach for system analysis as well as our conception of probabilistic/quantitative information in models.

References


