Non-determinism and Probabilities in Timed Concurrent Constraint Programming (Extended Version)

Jorge A. Pérez and Camilo Rueda

January 26, 2009

Abstract

We set the grounds of a framework for modeling and verifying systems involving probabilities, time, and partial information as constraints. A timed concurrent constraint process calculus with probabilistic and non-deterministic choices is proposed as description language. An operational semantics ensures consistent interactions between both kinds of choices. The semantics is also shown to be indispensable for the definition of logic-based verification capabilities over system specifications. The notion of *observable behavior* of a process –what an environment can perceive from its execution– is shown to provide a unified basis for modeling and reasoning about systems. We illustrate our approach by modeling and verifying a music improvisation scenario.

1 Introduction

This paper studies the integration of probabilistic information into a concurrent constraint process language with explicit discrete time and non-deterministic behavior. *Concurrent constraint programming* (ccp) [23, 16] is a model for concurrency with strong ties to logic. In ccp, systems are described by *constraints*, pieces of partial information that might include explicit quantitative parameters (such as, e.g., $x \leq 42$). Processes interact in a shared *store*; they either add new constraints or synchronize on the already available information. A distinguishing feature of ccp is its dual nature, operational and logic: processes can be seen, at the same time, as computing agents and logic formulas. This not only constitutes a rather elegant approach for verification; it is fair to say that ccp provides a unified framework for system analysis.

One possible critique to ccp is that it is too generic for representing certain complex systems. Even if counting with partial information is extremely valuable, we find that properly taking into account certain phenomena is cumbersome, which severely affects both modeling and verification. Particularly challenging is the case of *uncertain behavior*. Indeed, in areas such as, e.g., computer music [22] and systems biology [10, 2], the uncertainty underlying concurrent interactions goes way beyond what can be modeled using partial information only. Uncertain behavior is usually accompanied by manifestations of time, non-determinism, and asynchrony. One important commonality to many systems featuring uncertain behavior is that they can be described *probabilistically.* Indeed, apart from providing an intuitive way of describing alternative behaviors, probability distributions often ease the integration of statistic and empirical data into models.

We believe that starting from ccp-based languages for the analysis of specialized systems involving probabilistic behavior is reasonable. In fact, introducing suitable forms of probabilistic behavior in ccp could enhance the accuracy of specifications (since, e.g., more empirical information could be considered) and give more significance to verification, as provable properties would involve explicit probabilistic information.

We shall advocate the need of explicitly including time and non-determinism in our ccp-based description language. As for *time*, given its importance in reactive systems, we shall restrict ourselves to the realm of *timed ccp* (tccp) calculi [24]; hence, we aim at counting with explicit time and probability information in the modeling and verification phases. *Non-determinism* is of the essence to achieve compositionality and to abstract away from unimportant details. Furthermore, we think it is fundamental to faithfully represent the interactions of a system with its environment. We argue that replacing non-determinism with probabilistic choices is unsatisfactory for modeling purposes; such a decision entails making strong assumptions regarding interactions and/or specific conditions that are usually unknown or very hard to predict. Instead, a more convenient approach is to consider probabilistic systems within *non-deterministic environments*. As an example, consider the analysis of anonymity in security protocols [19]. There, a protocol (the reactive system) can be well described probabilistically, whereas the protocol users (the environment) are non-deterministic as their intentions are changing and the frequency of their interactions inherently unpredictable. The advantages of non-determinism in conjunction with probabilities usually come at the price of a more involved semantic support, aimed at handling both choices consistently.

Here we 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* process specifications in terms of properties including explicit quantitative information. Our proposal consists of the following key ideas:

- *A timed language with both probabilistic and non-deterministic behavior.* We define `pntcc`, a tccp calculus featuring non-deterministic and probabilistic choices. The discrete time in `pntcc` (introduced in Sect. 3) allows to relate it to languages such as `ntcc` [15] and `tcc` [24]. Remarkably, `pntcc` allows to naturally derive a notion of *probabilistic eventuality* that represents the eventual execution of processes based on quantitative parameters. As hinted at above, interactions among probabilistic and non-deterministic choices can be subtle and difficult to predict; a careless definition of their meaning could easily lead to inconsistencies. We thus equip `pntcc` with an operational semantics that (1) ensures consistent interactions between both kinds of choices, and by doing so (2) sets the ground for performing model checking processes (see below). The semantics is based on a *probabilistic automaton* [26] that separates the *internal* choices made probabilistically by the process from those *external* choices made non-deterministically under the influence of a *scheduler*. As a result, the *observable behavior* of a system —what the environment perceives from its execution— formalized by the semantics is purely probabilistic; the influence of non-determinism is regarded as unobservable.

- *A straightforward connection with model-checking procedures.* The rationale given by the language and the support provided by operational semantics make it possible a natural relation between `pntcc` and the probabilistic logic PCTL [11]. Formulas in PCTL express *soft deadlines*, statements explicitly involving a time bound and a probability. Hence, they allow for including quantitative parameters directly in the properties of interest. The relation between `pntcc` and PCTL (introduced in Sect. 4) is based on the fact that the observable behavior of a process can be interpreted as the discrete time Markov chain (DTMC) defining satisfaction in PCTL. This way, model checking for tccp process specifications becomes possible.

We believe that it is in the conception of process verification where our proposal differs from similar ccp calculi in which explicit time is considered [15, 17], which rely on proof systems for verification (see Sect. 6 for a review of related work.) We are not aware of other explicitly timed ccp calculi with quantitative parameters in both models and properties, and with the possibility of model-checking procedures. Our work provides the necessary foundations for a tccp-based *framework* for the analysis of reactive systems. In fact, since we advocate a rationale in which verification is an orthogonal concern, we can safely rely on the developed field of probabilistic model checking for this. As a matter of fact, we have already built a prototype that integrates a simulation tool for tccp processes [3] with the probabilistic model checker PRISM [14, 18]. In Sect. 5 we put our approach at work in an application that is representative of the notion of probabilistic systems/non-deterministic environments discussed before. We propose a `pntcc` model of a music improvisation reactive system interacting with a human musician, and show how a soft deadline can be verified with the aid of PRISM.

Some readers might wonder why we have started from a tccp calculus to define a description language if, after all, specifications are exported to a standard model-checking model. The crucial observation is that in our approach the states of the model involve partial information as declarative constraints. We find it convenient to export process specifications involving partial and probabilistic information to a verification setting that is well-understood, both in theory and in practice.

All in all, by relying on well-established techniques from logic and concurrency theory, this work sets the ground for new techniques and tools for the analysis of reactive systems involving quantitative and partial information.

2 Preliminaries

We introduce some necessary concepts from probability theory and a description of the Probabilistic Automata model.

Probability Theory. Given a set C, a σ-field on C (denoted by $\mathcal{F}$), is a subset of 2^C that is closed under complement and countable union and includes the empty set. The pair $(C, \mathcal{F})$ is called a *measurable space*. A *measure* μ over a measurable space $(C, \mathcal{F})$ is a function that assigns a non-negative real-value (possibly ∞) to each element of $\mathcal{F}$, such that (i) $\mu(\emptyset) = 0$ and (ii) if $(C_i)_{i \in N}$ is a sequence of pairwise disjoint elements of $\mathcal{F}$, then $\mu(\cup_i C_i) = \sum_i \mu(C_i)$. If $\mu(C) = 1$, then μ is called a *probability measure* or *probability distribution*.

A *probability space* is a triple $(C, \mathcal{F}, P)$, where $(C, \mathcal{F})$ is a measurable space, and P is a probability measure. The set C is called the *sample space*. We use $\mathcal{P}$ (possibly decorated) to denote a generic probabilistic space. This way, e.g., $\mathcal{P}'$ represents the probabilistic space $(C', \mathcal{F}', P')$. The set of probabilistic spaces over the set C is denoted $Probs(C)$.

Probabilistic Automata. Probabilistic automata [26] provide a unified model for analyzing systems involving both non-deterministic and probabilistic choices. A *probabilistic automaton* (PA in the sequel) is defined as a tuple $(Q, \overline{q}, A, T)$, where Q is a countable set of *states*, $\overline{q} \in Q$ is the *start state*, A is a countable set of *actions*, and $T \subseteq Q \times Probs(A \times Q)$ defines the *transition groups* of the automaton. Hence, a PA differs from an ordinary one only in the transition relation defined by T. For each state q_i, 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.

Probabilistic automata can be either *simple* or *fully probabilistic*. In a *simple* PA for each transition group $(q, \mathcal{P}) \in T$ there is an action $a \in A$ such that $C \subseteq \{a\} \times Q$, i.e., each transition is associated with a single action. Once a transition group is chosen, then only the next state is chosen probabilistically. A PA 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. In other words, a fully probabilistic automaton does not contain any non-determinism.

Given a simple PA, a *scheduler* (or *adversary*) resolves the non-determinism in the automaton based on the history of its execution. More precisely, in a (simple) PA M in the state q, a scheduler will select a specific transition group $(q, \mathcal{P}_i)$ from $(q, Probs(A \times Q))$, the set of all transition groups starting in q. This selection is usually left unspecified to model any scheduling policy.

3 A probabilistic, timed ccp process language

Here we describe the syntax and operational semantics for `pntcc`. We begin by introducing some notions of (explicitly timed) ccp-based calculi.

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

Discrete Time. Time is assumed to be divided into *units* (or *intervals*). In a given time unit, a process P gets an input (a constraint) c from the environment, it executes with this input as the initial *store* and when it reaches its resting point it *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 Process Syntax

Processes $P, Q, \ldots \in Proc$ are built from constraints $c \in \mathcal{C}$ and variables $x \in \mathcal{V}$ in some underlying constraint system. We start by giving the syntax of a "core" timed ccp [24]:

$$P, Q \quad ::= \quad \mathsf{skip} \;\Big|\; \mathsf{tell}(c) \;\Big|\; \mathsf{when}\, c\, \mathsf{do}\; P \;\Big|\; P \parallel Q \;\Big|\; \mathsf{local}\; x \;\mathsf{in}\; P \qquad (1)$$
$$\Big|\; !P \;\Big|\; \mathsf{next}\;(P) \;\Big|\; \mathsf{unless}\; c\; \mathsf{next}\;(P) \qquad (2)$$

Constructs in line (1) describe untimed ccp processes, whose action takes place during a single time unit. skip does nothing. $\mathsf{tell}(c)$ adds constraint c to the current store, thus making it available to other processes. $\mathsf{when}\, c\, \mathsf{do}\; P$ (a *positive ask*) checks if the current store is strong enough to entail the guard c; if so, it behaves like P. Given a finite set of indices I, process $\sum_{i \in I} \mathsf{when}\; c_i\; \mathsf{do}\; P_i$ generalizes positive asks as a *non-deterministic choice*: a process P_j $(j \in I)$ whose guard c_j is entailed from the current store is scheduled for execution; the chosen process precludes the others. The parallel composition $P \parallel Q$ describes the concurrent operation of P and Q, possibly "communicating" via the common store. Hiding on a variable x is enforced by process $\mathsf{local}\; x\; \mathsf{in}\; P$: it behaves like P, except that all the information on the x produced by P can only be seen by P and the information on x produced by other processes cannot be seen by P. We sometimes abbreviate $\mathsf{local}\; x_1\; \mathsf{in} \ldots \mathsf{local}\; x_n\; \mathsf{in} P$ (i.e. hiding on a list of pairwise distinct variables) as $\mathsf{local}\; x_1, \ldots, x_n\; \mathsf{in}\; P$.

Constructs in line (2) allow the above processes to have effect along the time units. Process $\mathsf{next}\;(P)$ schedules P for execution in the next time unit; it is thus a one-unit delay. Process $\mathsf{unless}\; c\; \mathsf{next}\;(P)$ is similar: P will be activated only if c cannot be inferred from the current store. Unless processes can be seen as (weak) time-outs: they 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. The *replication* operator ! represents infinite behavior: $!P$ represents $P \parallel \mathsf{next}\;(P) \parallel \mathsf{next}^2 P \parallel \ldots$, i.e. unboundedly many copies of P but one at a time.

The syntax of pntcc extends that of core timed ccp processes as follows.

Definition 1 *The syntax of* pntcc *processes is obtained by extending that of core tccp processes with the* probabilistic choice operator

$$\bigoplus_{i \in I} \mathsf{when}\; c_i\; \mathsf{do}\; (P_i, a_i),$$

where I is a finite set of indices, and for every $a_i \in \mathbb{R}^{(0,1]}$ we have $\sum_{i \in I} a_i = 1$.

The intuition of this operator is as follows. Each a_i associated with process P_i represents its probability of being selected for execution. Hence, the collection of all a_i represents a probability distribution. The guards that can be entailed from the current store determine a subset of *enabled processes*, which are used to determine an eventual normalization of the a_is. In the current time interval, the summation probabilistically chooses one of the enabled process according to the distribution defined by the (possibly normalized) a_is. The chosen alternative, if any, precludes the others. If no choice is possible then the summation is precluded.

Some notation for non-deterministic and probabilistic choices follow. In both cases, empty sums are abbreviated as skip; binary sums are denoted by "$+$" and "$\oplus$", resp. We use $\sum_{i\in I} P_i$ and $\bigoplus_{i\in I}(P_i, a_i)$ as shorthands for "blind choice" processes of the form $\sum_{i\in I} \mathsf{when}\, \mathtt{true}\, \mathsf{do}\ P_i$ and $\bigoplus_{i\in I} \mathsf{when}\ \mathtt{true}\ \mathsf{do}\ (P_i, a_i)$, resp. Sometimes we omit coefficients a_i in probabilistic choices with a uniform probability distribution. Thus, e.g., a process $\mathsf{when}\, c\, \mathsf{do}\ (P, 0.5) \oplus \mathsf{when}\, d\, \mathsf{do}\ (Q, 0.5)$ is denoted $\mathsf{when}\, c\, \mathsf{do}\ P \oplus \mathsf{when}\, d\, \mathsf{do}\ Q$.

A structural congruence relation $\equiv$ identifies processes with minor syntactical differences. It is the same as in `ntcc` and `utcc` and we include it here for the sake of completeness. It is given by the axioms: (1) $P \parallel \mathsf{skip} \equiv P$; (2) $P \parallel Q \equiv Q \parallel P$; (3) $P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R$; (4) $\mathsf{local}\ x\ \mathsf{in}\ Q \equiv \mathsf{local}\ x\ \mathsf{in}\ (P \parallel Q)$ if x is not in the free variables of P; (5) $\mathsf{local}\ x\ \mathsf{in}\ \mathsf{skip} \equiv \mathsf{skip}$.

The following example gives an intuition of computation in `pntcc`. The intuition shall be better formalized later, when discussing the operational semantics.

Example 1 *Consider the process P below. In the presence of a certain resource X the system emits a signal $out = 1$ with probability q; in case some additional component Y is also present, it emits the signal $out = 0$ (with probability $1 - q$).*

$$P = \mathsf{when}\, X > 0\, \mathsf{do}\ (\mathsf{tell}(\mathit{out} = 1), q)\ \oplus\ \mathsf{when}\, X > 0\ \wedge\ Y = 1\, \mathsf{do}\ (\mathsf{tell}(\mathit{out} = 0), 1 - q).$$

P evolves depending on the current store; we illustrate a few possibilities. The simplest one is the (empty) store `true`*: since there is no process enabled, no choice takes place and P is blocked.*

Say d, the store, is equal to $d = (X \geq 50)$, P causes the next store to become $d \wedge (\mathit{out} = 1)$ with probability 1. This occurs as $(X \geq 50) \models (X > 0)$ and a renormalization of the probabilities takes place (the guard of the second alternative cannot be entailed from d).

Given the store $e = (X = 20)\ \wedge\ (Y = 1)$: both guards in P are enabled, and the defined probabilistic choice can occur: the final store could be either $e \wedge (\mathit{out} = 1)$ with probability q, or $e \wedge (\mathit{out} = 0)$ with probability $1 - q$.

3.2 Operational Semantics

The operational semantics considers *configurations* of the form $\langle P, c\rangle$, where P is a process and c is a constraint representing a store. γ, γ' range over an element of the set of configurations Γ; γ_i denotes configuration $\langle P_i, c_i\rangle$. For configurations, we decree that $\langle P, c\rangle \equiv \langle Q, c\rangle$ iff $P \equiv Q$.

The semantics captures two levels of behavior as two different transition relations: 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. These internal transitions are defined as a probabilistic automaton. As such, there is an automaton (with its corresponding scheduler) for each time unit. By confining non-deterministic choices to internal computations, we obtain that only probabilistic behavior can be observed along time.

Internal Transitions. For each time unit i, with starting configuration γ_i, there is a PA $M_i = (\Gamma_i, \gamma_i, T_{M_i})$. The set $\Gamma_i \subseteq \Gamma$ is given by all those $\gamma_j \in \Gamma$ reachable from γ_i. The notion of action is not considered in this automaton; hence, the transitions of each

M_i are unlabeled. T_{M_i} denotes the transition groups of M_i; to denote each of them we borrow the following notation from [12]:

$$\gamma_i\{\longrightarrow_{p_i} \gamma_{i+1}\},$$

where $(\gamma_i, \mathcal{P}_i) \in T_{M_i}$ and $P_i(\gamma_{i+1}) = p_i$ (recall that $\mathcal{P}_i$ stands for the probabilistic space $(C_i, \mathcal{F}_i, P_i)$.) We use $\mathtt{tg}(\gamma_k)$ to denote the set of transition groups reachable from γ_k.

Rules in Table 1 define the transition groups in T_{M_i}. Rules TELL, UNL and REP formalize the informal descriptions for tell, unless and the replication processes given before; their associated probability is 1. A similar criterion applies for LOC and STR (that represent locality and structural congruence in reductions, resp.); their associated probability p_i is taken from a previous transition. For operational reasons, in rule LOC we use notation local (x, c) in P to express that c is the (store of) local information produced by a process local x in P. Rule PSUM formalizes probabilistic choices. The probability associated with such a choice (a'_j) is defined as

$$a'_j = \frac{a_j}{\sum_{k \in \{j \in I \mid d \models c_j\}} a_k}, \tag{3}$$

where d is the current store and a_j is the probability of the enabled process chosen, which is normalized taking into account the probabilities associated with the enabled processes.

An explanation on the rôle of non-determinism in the rules of Table 1 is worthwhile. Rule PAR formalizes parallel composition, allowing the interleaving of parallel processes inside transition groups. The involved processes maintain their transitions. The seemingly missing case for non-deterministic choice is covered by the scheduler of the PA. As parallel compositions, non-deterministic choices can occur inside transition groups, but they are under the influence of the scheduler. As such, no rule as in Table 1 is necessary for them.

We now state some basic conditions for a scheduler.

Definition 2 *Given a PA M_i, its associated* scheduler *is a function $Sch_i : T_{M_i} \to T_{M_i}$ that solves the non-deterministic choices present in a state $\gamma_j = \langle P_j, c_j \rangle$, possibly by taking into account the information given by c_j.*

In what follows, we use S, S' to range over schedulers as in Def. 2. One could expect that given a state with a non-deterministic choice, a scheduler first determines a set of enabled processes (those whose guards are entailed from the store of the given state) and then enforces some scheduling policy. Consequently, the ability of using the information in the store is naturally justified for a scheduler.

At this point, we find it convenient to describe the sequence of computations that occurs *within* a time unit. Since we consider both non-deterministic and probabilistic choices, we expect such a sequence to be under the influence of both the scheduler and the probabilistic choices defined by the PA. Even if there can be many sequences because of the choices involved, the environment can only observe one of them. We thus define an *internal sequence* of transitions as follows.

Definition 3 *Suppose a PA M_i with scheduler S_i. Notation $\gamma_1\{\longrightarrow_b^* \gamma_n\}_{S_i}$ represents the* internal sequence

$$\gamma_1 \longrightarrow_{a_1} \gamma_2 \quad \cdots \quad \gamma_{n-1} \longrightarrow_{a_{n-1}} \gamma_n$$

where $b = a_1 \times \cdots \times a_{n-1}$, and for every $k \in \{1, \ldots, n-1\}$, $S_i(\mathsf{tg}(\gamma_k)) = (\gamma_k, \mathcal{P}_{k+1})$ (non-deterministic choice), $P_k(\gamma_{k+1}) = a_k$ (probabilistic choice).

Note that the explicit reference to the scheduler in the internal sequence does not imply that a fixed scheduler is assumed per time unit. Rather, S_i denotes one of the possible schedulers that could be associated with M_i.

Observable Transition. An observable transition assumes a particular internal sequence (Def. 3) leading to a state where no further computation is possible (*quiescence*, denoted by $\not\longrightarrow$).

Definition 4 *Given an internal sequence starting in $\langle P, c\rangle$, finishing at $\langle Q, d\rangle$ and governed by a scheduler S_j, rule* OBS *below defines an* evolution *from process P.*

$$\textsc{Obs}\quad \frac{\langle P, c\rangle\{\longrightarrow_a^* \langle Q, d\rangle\}_{S_j} \not\longrightarrow}{P \xRightarrow{\langle c,d,a\rangle}_{S_j} F(Q)}$$

where $F(Q)$ represents the "future" of process Q. Let $F : Proc \rightharpoonup Proc$ be defined by

$$F(Q) = \begin{cases} \mathsf{skip} & \text{if } Q = \mathsf{skip} \text{ or } Q = \bigoplus_{i\in I} \mathsf{when}\ c_i\ \mathsf{do}\ (Q_i, a_i) \text{ or } Q = \sum_{i\in I} \mathsf{when}\ c_i\ \mathsf{do}\ Q_i \\ F(Q_1) \parallel F(Q_2) & \text{if } Q = Q_1 \parallel Q_2 \\ \mathsf{local}\ x\ \mathsf{in}\ F(R) & \text{if } Q = \mathsf{local}\ (x, c)\ \mathsf{in}\ R \\ R & \text{if } Q = \mathsf{next}\ (R) \text{ or } Q = \mathsf{unless}\ c\ \mathsf{next}\ (R) \end{cases}$$

Intuitively, $F(Q)$ 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.

The explicit reference to the scheduler in the observable transition becomes relevant when one considers that what can be observed from a process execution might differ depending on the particular scheduler. For simplicity, we sometimes omit it when the scheduler is not an issue or when it can be inferred from the context.

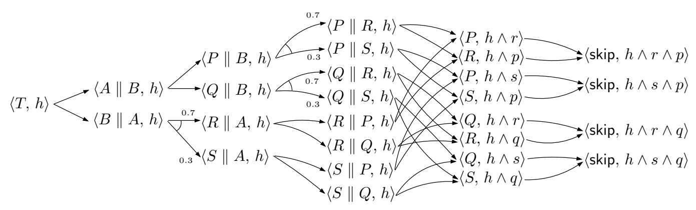
The following example illustrates the operational semantics and its associated notions.

Example 2 *Consider non-deterministic and probabilistic choices in processes A and B defined as*

$$A \overset{def}{=} \mathsf{when}\ c\ \mathsf{do}\ P + \mathsf{when}\ d\ \mathsf{do}\ Q + \mathsf{when}\ d'\ \mathsf{do}\ U \quad B \overset{def}{=} \mathsf{when}\ e\ \mathsf{do}\ (R, 0.7) \oplus \mathsf{when}\ e'\ \mathsf{do}\ (S, 0.3)$$

and let $T \overset{def}{=} A \parallel B$. Suppose an initial store h which entails all the guards but d'. Further, suppose that $P = \mathsf{tell}(p)$, $Q = \mathsf{tell}(q)$, $R = \mathsf{tell}(r)$, and $S = \mathsf{tell}(s)$. Starting from $\langle T, h\rangle$, we have a PA as in Figure 1. Initially there is a non-deterministic choice on the side of the composition to execute first. A scheduler could execute first the probabilistic choice of B and then the non-deterministic choice A, or viceversa. (Notice that because of the condition on h, process U is not part of the PA.) After that, the scheduler must deal only with non-determinism of the parallel composition. Given two different schedulers S_1 and S_2, two possible observable transitions are

$$T \xRightarrow{\langle h, h\wedge q\wedge r, 0.7\rangle}_{S_1} \mathsf{skip} \quad \text{and} \quad T \xRightarrow{\langle h, h\wedge p\wedge s, 0.3\rangle}_{S_2} \mathsf{skip}.$$



$\langle T, h\rangle$
$\langle A \parallel B, h\rangle$
$\langle B \parallel A, h\rangle$
$\langle P \parallel B, h\rangle$
$\langle Q \parallel B, h\rangle$
$\langle R \parallel A, h\rangle$
$\langle S \parallel A, h\rangle$
0.7
0.3
$\langle P \parallel R, h\rangle$
$\langle P \parallel S, h\rangle$
$\langle Q \parallel R, h\rangle$
$\langle Q \parallel S, h\rangle$
$\langle R \parallel P, h\rangle$
$\langle R \parallel Q, h\rangle$
$\langle S \parallel P, h\rangle$
$\langle S \parallel Q, h\rangle$
$\langle P, h \wedge r\rangle$
$\langle R, h \wedge p\rangle$
$\langle P, h \wedge s\rangle$
$\langle S, h \wedge p\rangle$
$\langle Q, h \wedge r\rangle$
$\langle R, h \wedge q\rangle$
$\langle Q, h \wedge s\rangle$
$\langle S, h \wedge q\rangle$
$\langle \mathsf{skip}, h \wedge r \wedge p\rangle$
$\langle \mathsf{skip}, h \wedge s \wedge p\rangle$
$\langle \mathsf{skip}, h \wedge r \wedge q\rangle$
$\langle \mathsf{skip}, h \wedge s \wedge q\rangle$

Figure 1: PA for the process T in Example 2. Semicircles denote probabilistic choices.

$$\textsc{Tell}\ \frac{}{\langle \mathsf{tell}(d), c\rangle\{\longrightarrow_1 \langle \mathsf{skip}, c \wedge d\rangle\}} \qquad \textsc{Par}\ \frac{\langle P, c\rangle\{\longrightarrow_{p_i} \langle P', c'\rangle\}}{\langle P \parallel Q, c\rangle\{\longrightarrow_{p_i} \langle P' \parallel Q, c'\rangle\}}$$

$$\textsc{PSum}\ \frac{}{\langle \bigoplus_{i\in I} \mathsf{when}\ c_i\ \mathsf{do}\ (P_i, a_i), d\rangle\{\longrightarrow_{a'_j} \langle P_j, d\rangle\}}\ \text{if } d \models c_j, j \in I.\ a'_j \text{ as in } (3).$$

$$\textsc{Unl}\ \frac{}{\langle \mathsf{unless}\ c\ \mathsf{next}\ (P), d\rangle\{\longrightarrow_1 \langle \mathsf{skip}, d\rangle\}}\ \text{if } d \models c \qquad \textsc{Str}\ \frac{\gamma_1\{\longrightarrow_{p_i} \gamma_2\}}{\gamma'_1\{\longrightarrow_{p_i} \gamma'_2\}}\ \text{if} \gamma_i \equiv \gamma'_i\ (i \in \{1,2\})$$

$$\textsc{Rep}\ \frac{}{\langle !P, c\rangle\{\longrightarrow_1 \langle P \parallel \mathsf{next}\ !P, c\rangle\}} \qquad \textsc{Loc}\ \frac{\langle P, c \wedge \exists_x d\rangle\{\longrightarrow_{p_i} \langle P', c'\rangle\}}{\langle \mathsf{local}\ (x, c)\ \mathsf{in}\ P, d\rangle\{\longrightarrow_{p_i} \langle \mathsf{local}\ (x, c)\ \mathsf{in}\ P', d \wedge \exists_x c'\rangle\}}$$

Table 1: Internal transition rules for `pntcc`.

3.3 Expressiveness of the Language: Derivable Operators

Probabilistic Eventuality. The probabilistic choice operator is expressive enough to define a parameterizable form of *probabilistic eventuality*, similar to the *unbounded eventuality* operator available in `ntcc` [15]. This is done by adapting the encoding of recursive definitions given in [15]. Such an encoding considers recursion definitions of the form $q(x) \stackrel{\text{def}}{=} P_q$. Here, q is the process name and P_q calls q within the scope of a "next" (this avoids infinite recursive calls within the same time interval). We assume recursion to be defined over a finite set of values V. Furthermore, the encoding presupposes two variables q, q_{arg} —not in the free variables of P_q. Notation $x \leftarrow t$ represents process $\sum_{v\in V}$ when $t = v$ do tell$(x = v)$, which defines the assignment of t's fixed value to x. Furthermore, let $(\!|P|\!)$ denote the process obtained by replacing in P any call $q(t)$ with tell$(call(q)) \parallel$ tell$(q_{arg} = t)$. This replacement is intended to signal a call of q with argument t and trigger a new invocation of P_q. The encoding of recursive definitions, denoted as $(\!|q(x) \stackrel{\text{def}}{=} P_q|\!)$ is given by

$$!(\mathsf{when}\ call(q)\ \mathsf{do}\ \mathsf{local}\ x\ \mathsf{in}\ (x \leftarrow q_{arg} \parallel (\!|P_q|\!))).$$

Whenever the process q is called with argument q_{arg}, the local x is assigned the argument's value so it can be used by q's body P_q. The encoding of calls $q(t)$ in other processes –denoted by $(\!|q(t)|\!)$– is defined by

$$\mathsf{local}\ q, q_{arc}\ \mathsf{in}\ ((\!|q(x) \stackrel{\text{def}}{=} P_q|\!) \parallel \mathsf{tell}(call(q)) \parallel \mathsf{tell}(q_{arg} = t)).$$

In `ntcc`, *eventuality* (or unbounded asynchrony) defines the eventual (but certain) execution of P in an undetermined future time unit. Sometimes, however, this could be a rather weak specification, especially if one counts with additional information regarding the eventual execution of a process. For instance, suppose a system in which some component is doomed to fail. Very often, in addition to knowing that the failure will indeed occur, it is reasonable to assume that the possibility of this failure changes as time advances. In fact, many processes could fit in a general scenario, in which their eventual execution is influenced by the passage of time.

To correct this situation, we propose a *probabilistic eventuality* operator that integrates the partial information on process occurrence as probability distributions. It combines probabilistic choices and recursion to express user-controlled delays:

$$\textsc{Star}_f(P, r) \quad \stackrel{\text{def}}{=} \quad (P, r) \oplus (\mathsf{next}\ \textsc{Star}_f(P, f(r)), 1 - r). \tag{4}$$

We now give an intuition on the definition of $\textsc{Star}_f(P, r)$, pointing out some relevant issues. We shall assume the operator is defined over a sufficiently large (yet finite) subset of the reals in $(0, 1]$, denoted by $\mathbb{R}^{(0,1]}$. The operator depends on two parameters, r and f. The first stands for the *current* probability of executing P: the closer to 1 r is, the greater the probability of executing P will be. Conversely, $1 - r$ denotes the probability of *delaying* P's execution. The fixed function $f : \mathbb{R}^{(0,1]} \to \mathbb{R}^{(0,1]}$ governs the execution of P by modifying r in each recursive call. The selection of appropriate parameters for f and r is the basis for describing diverse patterns of behavior based on (finite) recursions. The following example should make the rôle of parameters clearer.

Example 3 *Assume a constraint system over (a finite subset of) the reals extended with the predicate symbols* `fail`, `stop`, `go_on`. *Also, suppose a function g defined as $g(x) = min(1.0, x + 0.02)$. Let* $\mathrm{S} \stackrel{\text{def}}{=} \mathrm{C} \parallel \mathrm{R}$ *be a reactive system with processes* C *and* R *described below.*

$\mathrm{C} \stackrel{\text{def}}{=} \textsc{Star}_g(\mathsf{tell}(\mathtt{fail}), 0.7)$ $\quad \mathrm{R} \stackrel{\text{def}}{=} !\,(\mathsf{when}\ \mathtt{fail}\ \mathsf{do}\ \mathsf{next}\ (\mathsf{tell}(\mathtt{stop})) \parallel \mathsf{unless}\ \mathtt{fail}\ \mathsf{next}\ (\mathsf{tell}(\mathtt{go\_on})))$.

The parameters of the probabilistic eventuality in C *ensure that* S *will surely stop within 15 time units, with the probability of a fail increasing linearly. This way, e.g., if the system does not fail in the first time unit, in the following one the probability of fail will increase to 0.72.*

Notice that the definition of the probabilistic eventuality suggests other properties f should have. For instance, in some cases it could be desirable that f be monotonically (strictly) decreasing. To avoid being too specific, we prefer not to give a rigid set of conditions for f.

Other Operators. In the context of `pntcc`, a natural derived construct is the one for probabilistic execution, i.e., a construct that relies on probabilities to decide whether to schedule a process in the current time unit or not. For this purpose, consider the process $\textsc{Rho}(P, k)$ below, where P is a process and k is a quantitative parameter related with the execution of P in the current time unit.The execution of P is easily seen to depend on the value of parameter k:

$$\textsc{Rho}(P, k) \stackrel{\text{def}}{=} (P, k) \oplus (\mathsf{skip}, 1 - k). \tag{5}$$

4 Observable Behavior for the Verification of Processes

Here we relate `pntcc` processes and the temporal logic PCTL. This is possible because the observable behavior of a process that is given by the semantics provides a formal way of resolving any non-determinism. We show that such an observable behavior corresponds to the discrete time Markov chain (DTMC, in what follows) upon which satisfaction in PCTL is defined. In other words, we interpret observable behavior as satisfaction in PCTL. We first summarize the syntax and semantics of PCTL. Then, we give details on the interpretation.

4.1 The temporal logic PCTL

PCTL allows to reason about properties such as “after a request, a task will be accomplished within 5 minutes with a probability of at least 95%”. Such statements, so-called *soft deadlines*, explicitly define both a probability and a time bound. Unlike *hard* deadlines, soft deadlines are meant to characterize systems in which a failure does not imply catastrophic consequences. Next we briefly introduce PCTL, and refer the reader to [11] for further details.

Syntax and Expressivity. PCTL formulas 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. The former represent properties of states; the latter represent properties of sequences of states. More precisely:

- Every atomic proposition (denoted by a_i) is a state formula;
- If f_1 and f_2 are state formulas, then $\neg f_1$, $(f_1 \wedge f_2)$, $(f_1 \vee f_2)$, $(f_1 \rightarrow f_2)$ are also state formulas;
- Given two state formulas f_1 and f_2 and a non-negative number t (or ∞), $(f_1\, U^{\leq t}\, f_2)$ and $(f_1\, \mathcal{U}^{\leq t}\, f_2)$ 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 $\mathcal{U}$ are the *strong* and *weak* until operators, resp. 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 $[f_1\, \mathcal{U}^{\leq t}\, f_2]_{\geq p}$ with $f_1\, U^{\leq t}_{\geq p}\, f_2$ and $f_1\, \mathcal{U}^{\leq t}_{\geq p}\, f_2$, resp.

The difference between strong and weak until operators is as follows: formula $f_1\, U^{\leq t}_{\geq p}\, f_2$ 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 $f_1\, \mathcal{U}^{\leq t}_{\geq p}\, f_2$ 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 $f_1\, U^{\leq t}_{>p}\, f_2$ and $f_1\, \mathcal{U}^{\leq t}_{>p}\, f_2$ have an analogous meaning.

Several properties can be expressed in PCTL by building on U and $\mathcal{U}$. For instance, to express that a property will hold *continuously* during a specific time frame bounded by t with a probability p, we can decree $G^{\leq t}_{\geq p}\, f \equiv f\, \mathcal{U}^{\leq t}_{\geq p}\, \texttt{false}$. Similarly, to express that the property will hold *sometimes* during t we have $F^{\leq t}_{\geq p}\, f \equiv \texttt{true}\, U^{\leq t}_{\geq p}\, f$.

Semantics. Formulas are interpreted over DTMCs which are, essentially, fully-probabilistic automata. In fact, notice that the only difference between them is the labeling function:

Definition 5 *A DTMC is a tuple* $\langle S, s^i, \mathcal{T}, L\rangle$*, where* S *is a finite set of* states *(with* $s^i \in S$ *as* initial state*);* $\mathcal{T} : S \times S \rightarrow [0,1]$ *is a* transition function*, such that* $\forall s \in S$*,* $\sum_{s' \in S} \mathcal{T}(s, s') = 1$*; and the* labeling function $L : S \rightarrow 2^A$ *assigns atomic propositions to states.*

In a DTMC each transition is considered to require one time unit. Structures are represented as diagrams, where states and transitions are depicted as nodes and as directed arcs labeled with their associated probabilities, resp. Given a state s_0 of a structure, the infinite sequence $s_0, \ldots, s_n, \ldots$ is called a *path* σ from s_0. The i-th state of σ (s_i) is denoted by $\sigma[i]$. For each structure and state s_0, a measure μ_m on the set of paths from s_0 is defined on the measurable space $(X, \mathcal{A})$, where X is the set of paths starting in s_0 and $\mathcal{A}$ is a σ-field on X generated by sets of paths with a common finite prefix of length n. μ_m is uniquely defined on all sets of paths in $\mathcal{A}$.

Given a DTMC K, truth for formulas is defined by a satisfaction relation $s \models_K f$. A satisfaction relation over paths, denoted $\sigma \mathrel{|\!\!\equiv}_K f$, is also defined ($\sigma$ and f being a path and a path formula, resp.). These two relations, simply denoted as $\models$ and $\mathrel{|\!\!\equiv}$, are defined as follows:

$s \models a$	iff $a \in L(s)$
$s \models \neg f$	iff not $s \models f$
$s \models f_1 \wedge f_2$	iff $s \models f_1$ and $s \models f_2$
$s \models f_1 \vee f_2$	iff $s \models f_1$ or $s \models f_2$
$s \models f_1 \rightarrow f_2$	iff $s \models \neg f_1$ or $s \models f_2$
$s \mathrel{|\!\!\equiv} f_1\, U^{\leq t}\, f_2$	iff there exists an $i \leq t$ such that $\sigma[i] \models f_2$ and $\forall j : 0 \leq j < i : (\sigma[j] \models f_1)$
$s \mathrel{|\!\!\equiv} f_1\, \mathcal{U}^{\leq t}\, f_2$	iff $\sigma \mathrel{|\!\!\equiv} f_1\, U^{\leq t}\, f_2$ or $\forall j : 0 \leq j \leq t : (\sigma[j] \models f_1)$
$s \models [f]_{\geq (>)\, p}$	iff the μ_m-measure of the set of paths σ starting in s for which $\sigma \mathrel{|\!\!\equiv} f$ is at least (greater than) p
$s \models [f]_{\mathcal{R}p}$	iff the μ_m-measure of the set of paths σ starting in s for which $\sigma \mathrel{|\!\!\equiv} f$ is greater than p

Finally, it is defined that $\models f$ is equivalent to $s^i \models f$, where s^i is the initial state of the given structure.

4.2 Relating observable behavior and PCTL

Here we give the necessary definitions to go from the observable behavior of a process (as defined by rule OBS of the operational semantics) to the DTMC underlying satisfaction in PCTL (Def. 5). Since the non-determinism has been resolved all that is required is to give structure to the derivatives that can be observed from a process execution along time. We start by defining the set of such derivatives, henceforth called *alternatives*.

Definition 6 *Given a configuration $\gamma = \langle P, c\rangle$ and a scheduler S, the set of* alternatives of γ under S *is defined as* $\mathsf{alt}_S(\gamma) = \{\langle Q, d\rangle \mid P \xRightarrow{\langle c,d,a\rangle}_S Q\}$*, for any constraint d, and* $a \in (0, 1]$.

Alternatives give a one-step account of all the possibilities for observable behavior. Now it is straightforward to articulate the notion of *observable sequences* as the description of one of the possible computations possible starting in a given configuration. Notice that this is behavior *along the time units* rather than *within* a single time unit, as described by internal sequences (Def. 3).

Definition 7 *An* observable sequence *starting in γ_0 is defined as any sequence of configurations $s = \gamma_0, \gamma_1, \ldots, \gamma_n, \ldots$ such that for every $i \geq 0$, there exists $\gamma_{i+1} \in$* $\mathsf{alt}_{S_i}(\gamma_i)$*, S_i being the scheduler at time i.*

The observable sequences originating in a given process represent the confinement of non-deterministic behavior to the scheduler used over internal evolutions. This rôle of schedulers allows to interpret the observable behavior of a process as a DTMC. Indeed, given a process P, it is not difficult to think of a correspondence between *states* and all the possible configurations reachable from $\langle P, \texttt{true}\rangle$ through observable sequences; we will denote the set of such configurations as $Reach(P)$. *Transitions* of the automaton can be obtained from the *alternatives* of each derivative of P. Notice that, in this setting, an observable sequence (Def. 7) would then correspond to a particular path of the DTMC. Finally, we can assume a labeling function that labels states with the store of the given configuration.

Definition 8 *The DTMC associated with a* `pntcc` *configuration* $\gamma_o = \langle P_0, c_0\rangle$*, denoted* $\mathrm{DTMC}(\gamma_0)$*, is defined as the tuple* $(Q_{\mathtt{OBS}}, \gamma_0, T_{\mathtt{OBS}}, L_M)$*, where*

- $Q_{\mathtt{OBS}} = Reach(P_0)$ *is the set of states, with* γ_0 *as inital state;*
- $T_{\mathtt{OBS}} = Q_{\mathtt{OBS}} \times Q_{\mathtt{OBS}} \to [0,1]$ *defines the transition relation. For all* $\gamma \in Q_{\mathtt{OBS}}$ *it holds that (i)* $(\gamma, \gamma') \in T_{\mathtt{OBS}}$ *if* $\gamma' \in \mathsf{alt}_{S_i}(\gamma)$ *and (ii)* $\sum_{\gamma' \in Q_{\mathtt{OBS}}} T_{\mathtt{OBS}}(\gamma, \gamma') = 1$*;*
- $L_M : Q_{\mathtt{OBS}} \longrightarrow \mathcal{C}$ *is a labeling function such that given a configuration* $\gamma_i = \langle P_i, c_i\rangle$*,* $L_M(\langle P_i, c_i\rangle) = c_i$*, for each* $\gamma \in Q_{\mathtt{OBS}}$*.*

It is now possible to show the following proposition.

Proposition 1 *Given a* `pntcc` *process* P_0*, for every* P_n *reachable from* P_0 *through an observable sequence, in the DTMC given by* $\mathrm{DTMC}(\langle P_0, \texttt{true}\rangle)$ *there exists a path from* $\langle P_0, \texttt{true}\rangle$ *to* $\langle P_n, d\rangle$*, for some constraint d.*

In this way, a simple (yet formal) relationship between `pntcc` and PCTL is completed. Notice that from this definition of DTMC building a fully-probabilistic automaton is rather straightforward. First, one should drop the labeling function L_M; then, one should associate a notion of *actions* to transitions. A natural choice for this is to consider the *inputs* the environment provided to the given process.

A legitimate question at this point concerns the construction of DTMCs in practice. As mentioned in the introduction, we already count with a prototype tool that links `pntcc` specifications with DTMC specifications as required by the PRISM model checker. In a nutshell, our prototype uses a simulation tool for timed ccp processes [3] which at runtime extracts an intermediate representation from a `pntcc` specification. From such a representation (which is essentially the desired DTMC, plus some other simulation-related information) it is then easy to write a model input file for PRISM. We are currently working on professional implementations of all these software components.

5 The approach at work: probabilistic music improvisation

Here we use `pntcc` to account for a model of music improvisation. We shall illustrate music improvisation as but one representative example of a probabilistic reactive system in an environment that is inherently non-deterministic. We shall discuss how this distinction is key to achieve realistic models and to prove meaningful properties over them.

Improvisation usually takes place during a musician's performance, and involves two phases. The first one (learning) applies machine learning methods to musical sequences in order to capture salient musical features and organize these features into a model. The second (simulation) browses the model in order to generate variant musical sequences that are *stylistically consistent* with the learned material. If both learning and simulation processes occur in real-time, in an interactive system where the computer "plays" with musicians, then *machine improvisation* is achieved.

In [21] we have proposed a tccp model for machine improvisation based on a *Factor Oracle automaton* [1] (FO in what follows). The learning phase of the model constructs a FO with the given sequence; improvisation takes place by navigating the automaton. Unlike similar models, in which learning and improvising occur sequentially, in [21] these phases may occur concurrently. Under the criterion of being stylistically consistent, choices are a sensible aspect of model navigation. Indeed, they entail a creativity trade-off: one looks for pieces that are as novel and creative as possible, but while preserving the stylistic parameters of the learned piece. One significant drawback of the model in [21] is that choices are non-deterministic only. As a result, there is no control on improvisation phases and certain creative choices are limited.

Here we correct this situation by refining the navigation part of the model in [21] with probabilistic choices. In the musical setting this refinement serves two main purposes:

- it allows to have a quantitative measure of the quality of an improvised sequence;
- it enhances the control the modeler has over the whole process.

The human player in the improvisation process is essentially an external, non-deterministic agent, since, e.g., he plays notes at unpredictable times. Hence, counting with both probabilistic and non-deterministic behavior is essential to achieve a realistic description of the system. Before describing the improved model we give a short description of the properties and construction of FOs. Later on, we discuss the verification of a soft deadline over the proposed model.

Factor Oracles. A FO is a succinct representation of all the factors of a sequence of symbols. Given a sequence $s = \sigma_1\sigma_2 \ldots \sigma_n$, its FO consists of states $0, 1, 2 \ldots n$. There is always a transition arrow (called *factor link*) labeled by symbol σ_i going from state $i - 1$ to state $i, 1 \leq i < n$. Depending on the structure of s, other arrows are added to the automaton. Some are directed from a state i to a state j, where $0 \leq i < j \leq n$. These are also factor links and are labeled by symbol σ_j. Some are directed "backwards", going from a state i to a state j, where $0 \leq j < i \leq n$. They are called *suffix links*, and bear no label. The factor links model a factor automaton, that is, every factor p in s corresponds to a unique factor link path labeled by p, starting in 0 and ending in some other state. Suffix links have an important property: a suffix link goes from i to j iff the longest repeated suffix of $s[1..i]$ is recognized in j. Suffix links thus connect repeated patterns of s.

The FO is learned on-line. For each new entering symbol σ_i, a new state i is added and an arrow from $i-1$ to i is created with label σ_i. Starting from $i-1$, the suffix links are iteratively followed backward, until a state is reached where a factor link with label σ_i originates (going to some state j), or until there are no more suffix links to follow. For each state met during this iteration, a new factor link labeled by σ_i is added from this state to i. Finally, a suffix link is added from i to the state j or to state 0 depending on which condition terminated the iteration. Let δ_{k,σ_i} denote the state reached by following *forward* arc σ_i from state k in the automaton, and let S_j be the state reached by following the *backward* link from state j. A FO for the sequence $s = ab$ would then have three states $0, 1, 2$, with forward links $\delta_{0,a} = 1, \delta_{0,b} = 2, \delta_{1,b} = 2$ and backward links $S_2 = 0, S_1 = 0$.

Navigating the oracle in order to generate variants is straightforward: starting in any state, following factor links generates a sequence of labeling symbols that are *repetitions* of portions of the learned sequence; following one suffix link followed by a factor link creates a *recombined pattern* sharing a common suffix with an existing pattern in the original sequence. In music, navigating the FO means an improvisation process in which the interest is to obtain stylistically consistent combinations of repetitions and recombined patterns of the learned sequences.

Music improvisation in `pntcc`**.** A FO-based model for music improvisation in `pntcc` is given in Table 2. We assume a system composed of three subsystems: learning (LEARN), improvisation (IMPROV) and playing (PLAYER), that execute concurrently. Here a symbol σ_i is a note (say a pitch). Three kinds of variables are used to represent the partially built FO. Variables f_k denote the set of labels of all currently existing factor links going forward from k. Variables S_i are suffix links from each state i, and variables $\delta_{i,\sigma}$ give the state reached from i by following a factor link labeled σ. This way, e.g., for the FO for $s = ab$: $f_0 = \{a, b\}, f_1 = \{b\}$, $S_1 = 0$, $S_2 = 0$, $\delta_{0,a} = 1$, $\delta_{0,b} = 2$, $\delta_{1,b} = 2$. Variable *out* represents a note output by the improviser. In the example we use notation wait c do P to abbreviate process $W_{(c,P)} \overset{def}{=} \textsf{when}\, c\, \textsf{do}\ P \parallel \textsf{unless}\ c\ \textsf{next}\ (W_{(c,P)})$.

The learning and improvisation phases can be done concurrently. They must proceed in such a way that improvisation always works on a completely built subgraph. This is easily accomplished by synchronizing on S_i. Indeed, when S_i is determined the subgraph up to state i has been completely built. Process ADD (in LEARN) keeps adding symbols to the automaton provided the previous one has already been added (S_{i-1} is determined) and the performer has already played beyond the currently known symbols ($go \geq i$). For space reasons we omit the definition of ADD; see [21] for details. Notice that synchronization is greatly simplified by the use of constraints. If at a given moment variable S_i has no value, when processes depending on it are blocked.

Process PLAYER represents a musician that plays some note p every once in a while. It non-deterministically chooses between playing now or postponing the decision to the next time unit. When playing is decided a further non-deterministic choice is performed to select some symbol p (the note) from the alphabet. This represents the act of playing. Process IMPROV(k) represents improvisation from state k of the FO. With probability a, it chooses whether to follow a backward link S_k and then output some symbol $\sigma \in f_{S_k}$ with probability b_σ (i.e. to actually improvise), or, with probability $1-a$, to output symbol σ_{k+1} (i.e. to stick with the original sequence). When $S_k = -1$ there is no other choice but to output symbol σ_{k+1}. Improvisation starts after n symbols have been produced by the player. The whole system is thus denoted SYSTEM$_n$.

Verifying the model. We verify a soft deadline of the model in Table 2, making a few assumptions:

- Improvisation starts after five symbols have been played/learned, i.e. we consider SYSTEM$_5$.

$$
\begin{array}{rcl}
\textsc{Player}_j & \overset{\text{def}}{=} & (\mathsf{tell}(go = j-1) \parallel \mathsf{next}\ (\textsc{Player}_j)) \ + \\
& & \displaystyle\sum_{p\in\Sigma} \mathsf{when}\ \mathtt{true}\ \mathsf{do}\ (\ !\ \mathsf{tell}(\sigma_j = p) \parallel \mathsf{tell}(go = j) \parallel \mathsf{next}\ (\textsc{Player}_{j+1})) \\
\textsc{Learn}_i & \overset{\text{def}}{=} & \mathsf{when}\ S_{i-1} \geq -1 \wedge go \geq i\ \mathsf{do}\ (\textsc{Add}_i \parallel \mathsf{next}\ (\textsc{Learn}_{i+1})) \parallel \\
& & \mathsf{unless}\ S_{i-1} \geq -1 \wedge go \geq i\ \mathsf{next}\ (\textsc{Learn}_i) \\
\textsc{Improv}(k) & \overset{\text{def}}{=} & \mathsf{wait}\ go \geq k\ \mathsf{do} \\
& & \quad \mathsf{when}\ S_k = -1\ \mathsf{do}\ \mathsf{next}\ (\mathsf{tell}(out = \sigma_{k+1}) \ \parallel\ \textsc{Improv}(k+1)) \ \parallel \\
& & \quad \mathsf{when}\ S_k \geq 0\ \mathsf{do}\ (\mathsf{next}\ (\displaystyle\bigoplus_{\sigma\in\Sigma} \mathsf{when}\ \sigma \in f_{S_k}\ \mathsf{do}\ (\mathsf{tell}(out = \sigma) \parallel \textsc{Improv}(\delta_{S_k,\sigma}), b_\sigma)),\ a) \\
& & \qquad \oplus \\
& & \quad \mathsf{when}\ S_k \geq 0\ \mathsf{do}\ (\mathsf{next}\ (\mathbf{tell}(out = \sigma_{k+1}) \parallel \textsc{Improv}(k+1)),\ 1-a) \\
\textsc{System}_n & \overset{\text{def}}{=} & !\ \mathsf{tell}(S_0 = -1) \parallel\ \textsc{Player}_1\ \parallel\ \textsc{Learn}_1\ \parallel\ \textsc{Improv}(n)
\end{array}
$$

Table 2: `pntcc` model for music improvisation.

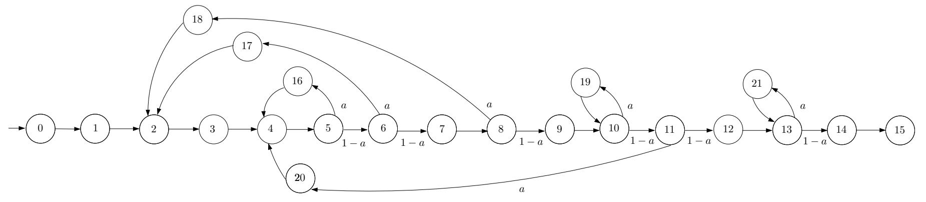


Figure 2: DTMC for the music improvisation process. Unlabeled arcs have probability 1.

- For simplicity, the scheduler that solves the non-determinism of the player never postpones playing a note. Therefore, the sequence will be performed continuously. The verification procedure to be presented is the same for more involved scheduler definitions.
- The player performs a sequence of J.S. Bach's *two-part Invention No. 5*, as studied in [5]. More precisely, we take the sequence $s = (0, 51, 63, 62, 63, 0, 65, 0, 67, 67, 63, 68, 68, 58, 60)$ corresponding to pitches (in MIDI notation, value 60 is the note middle C). The FO for such a sequence is given in Figure 3.

After setting these conditions, let us state the property of interest. It concerns the rate of repetition vs. improvisation one could obtain from the model. In the rest of this section, we shall be interested in the set of *improvisation states* of the model, i.e., all the configurations of the process $\textsc{System}_5$ just after scheduling a $\textsc{Improv}$ process. We denote such a set with $S_I = \{s_{i1}, \ldots, s_{in}\}$.

Proposition 2 *Given the model for music improvisation in Table 2, we then have that, within a time bound given by t, the system will reach an improvisation state with a probability p. In PCTL:*

$$f_1 = F^{\leq t}_{\geq p}\ s_{i1} \vee \cdots \vee s_{in}.$$

Our objective will be then to determine the value of the probability p. The first step in the verification is to determine the DTMC representing the behavior of $\textsc{System}_5$.

	Time units (t)							
Probability (a)	7	8	9	10	11	12	13	14
0.1	0.19	0.19	0.271	0.271	0.3439	0.40951	0.40951	0.46856
0.2	0.36	0.36	0.488	0.488	0.5904	0.67232	0.67232	0.73786
0.3	0.51	0.51	0.657	0.657	0.7599	0.83193	0.83193	0.88235
0.4	0.64	0.64	0.784	0.784	0.8704	0.92224	0.92224	0.95334
0.5	0.75	0.75	0.875	0.875	0.9375	0.96875	0.96875	0.98438
0.6	0.84	0.84	0.936	0.936	0.9744	0.98976	0.98976	0.9959
0.7	0.91	0.91	0.973	0.973	0.9919	0.99757	0.99757	0.99927

Table 3: Experiments with the `pntcc` model of music improvisation. Each cell gives a value for p in Proposition 1: the probability of reaching an improvisation state within t time units, when the model in Table 2 is instantiated with probability a in the IMPROV subprocess.

It is given in Figure 2. There, each configuration of the system in time is numbered. State 0 corresponds exactly to $\langle \text{SYSTEM}_5, \texttt{true} \rangle$; for every state $i > 0$, it holds that $go = i$. States 5, 6, 8, 10, 11, 13 represent the system when a improvisation choice is possible; in case improvisation is chosen, they evolve to states 16-21, resp. The FO for the sequence decrees that there is at most one suffix link for each state; hence, no probabilistic choice relying on b_σ takes place. Notice also that both the model in Table 2 and Proposition 2 are stated without fixed values for the probabilities of improvising (a_i) or the time bound t. This allows us to experiment with the model, selecting diverse parameters and observing their effect in the final probability (that is, p). Using the model checker PRISM [14, 18] this can be done in an automatic way. Table 3 reflects our results for a sample of parameters. Notice how departing from a very succinct model, we are able to obtain a very rich set of quantitative information. Even if the tendencies on the value of p are rather immediate to infer, it is worthwhile observing how taking $a = 0.7$ and $a = 0.6$ leads to a quick convergence towards an improvisation state. For lower values of probabilities, this is a less evident tendency, that becomes clear only when analyzing a longer time frame. Finally, one may notice that the values for some pairs of columns (7-8, 9-10, 12-13) are the same. This is due to the specific shape of the DTMC, which in turn, has to do with the suffix links in the underlying FO.

We conclude our example by observing that the model in Table 2 could be the basis for more sophisticated analysis. We believe this is a very convenient aspect, as the model is succinct and intuitive, and therefore only a few, precise modifications would be needed. One refinement idea would be to tailor probabilities so that larger contexts (i.e. repeated factors) are preferred. Also, analyzing several improvisation processes (say, the members of a orchestra) following the same human player appears challenging. We leave to a future contribution the thorough analysis of these issues.

6 Related Work

Work in [9] extends ccp (as in [25]) and timed ccp (`tcc` [24]) with discrete random variables, which are conceptually closer to the idea of *stochastic behavior* than to the notion of the probabilistic choice here proposed. Also, while in [9] the interest is on denotational semantics, here we have focused on the operational side of timed ccp. In [20] an untimed probabilistic ccp language and its operational semantics are proposed. The language —intended for the analysis of randomized algorithms— replaces non-deterministic choices with probabilistic ones. Consequently, the semantics provided in [20] could be seen as a particular case of ours.

In [7, 8] Falaschi et al. have addressed the model checking of timed ccp languages without probabilistic information. In [8] they propose model checking for the non-deterministic, timed ccp language in [6], which features a different notion of discrete time. Perhaps closer to our work is [7], which shows how to extract model checking structures directly from `tcc` [24] programs. Transitions in the extracted structures are rather ad-hoc, which prevents the use of classical model checking techniques over them. The explicit rôle of non-determinism in our proposal makes such an "extraction approach" unfeasible; we discussed how following the operational semantics of the language is a necessary step to derive a DTMC. In this sense, the DTMCs we derive are "operationally correct" by construction.

Work in [4] introduces sCCP, a stochastic ccp language targeted at biological systems. Being stochastic, sCCP provides a framework in which discrete and continuous time can be treated only *implicitly*. This is to be contrasted with the explicit rôle discrete time has in `pntcc`, which is prominent in the language constructs and in their semantics. The treatment of (stochastic) quantitative information in sCCP finds its motivation in biologic applications. In sCCP, ask and tell processes are endowed with rates (positive real numbers) that depend on the information in the store. Rates can be interpreted either as a *priority* within a probability distribution or as the *stochastic duration* of a process (by an association with random variables). An operational semantics that considers instantaneous and stochastic transitions is proposed. As in our work, [4] proposes model-checking of sCCP processes by an encoding into the input language of PRISM. Similarly to [7] (cited above), sCCP allows for the extraction of model checking directly from programs. We stress that our approach has to be different —more involved— in that we need an additional semantic support to be able to derive a DTMC, and in turn, to perform a model checking process.

7 Concluding Remarks

We have proposed the basis of a framework (a description language plus reasoning techniques) for analyzing reactive systems involving constraints, explicit time, probabilities and non-determinism. We introduced `pntcc`, a discrete-timed ccp language that combines non-deterministic and probabilistic choices. A distinctive construct in `pntcc` is the one expressing *probabilistic eventuality*, which describes the eventual execution of a process, possibly under the influence of complex patterns of behavior (which are representable as the parameters of the operator). The operational semantics for `pntcc` gives coherence to the two kinds of choices provided by the language, and is robust enough to support alternative definitions of (non-deterministic and probabilistic) choice operators. This allows for language extensions tailored to some particular application.

The pertinence of a new formalism can be well assessed by the reasoning techniques it provides. We have described how `pntcc` processes can be related with the probabilistic logic PCTL by analyzing their observable behavior. It was shown how the observable behavior of `pntcc` processes (formalized by the operational semantics) can be interpreted as the DTMC defining satisfaction in PCTL. Remarkably, this interpretation paves the way for an alternative approach for verification of reactive systems.

The rationale behind the design of `pntcc` is independent of any application domain. To demonstrate the usefulness of the approach, we analyzed a non-trivial problem in the setting of music improvisation. We provided an enhanced model in which the combination of non-determinism and probabilities is indispensable and, by exploiting the quantitative information in it, we performed a series of experiments on a PTCL property. Apart from applications in computer music, we think pntcc has a place in other areas, including "traditional" ones such as performance analysis (in the line of languages such as PEPA [13]) and emerging ones such as systems biology, an area in which timed ccp has already proven useful [10, 2].

References

[1] C. Allauzen, M. Crochemore, and M. Raffinot. Factor oracle: a new structure for pattern matching. In *Proc. of SOFSEM'99*, volume 1725 of *LNCS*. Springer, 1999.

[2] A. Arbeláez, J. Gutiérrez, and J. A. Pérez. Timed Concurrent Constraint Programming in Systems Biology. *Newsletter of the ALP*, 19(4), November 2006.

[3] AVISPA Research Group. ntccSim: A simulation tool for timed concurrent processes, 2008. `http://cic.puj.edu.co/wiki/doku.php?id=grupos:avispa:ntccsim`.

[4] L. Bortolussi. *Constraint-based approaches to stochastic dynamics of biological systems*. PhD thesis, University of Udine, 2007.

[5] A. Cont, S. Dubnov, and G. Assayag. Anticipatory model of musical style imitation using collaborative and competitive reinforcement learning. In *Anticipatory Behavior in Adaptive Learning Systems*, volume 4520 of *LNAI*, pages 285–306. Springer Verlag, Berlin, 2007.

[6] F. S. de Boer, M. Gabbrielli, and M. C. Meo. A timed concurrent constraint language. *Inf. Comput.*, 161(1):45–83, 2000.

[7] M. Falaschi, A. Policriti, and A. Villanueva. Modeling concurrent systems specified in a temporal concurrent constraint language-i. *Electr. Notes Theor. Comput. Sci.*, 48:197–210, 2001.

[8] M. Falaschi and A. Villanueva. Automatic Verification of Timed Concurrent Constraint Programs. *Theory and Practice of Logic Programming*, 6(3):265–300, May 2006.

[9] V. Gupta, R. Jagadeesan, and V. Saraswat. Probabilistic concurrent constraint programming. In *CONCUR '97*, volume 1243 of *LNCS*, pages 243–257. Springer-Verlag, 1997.

[10] J. Gutiérrez, J. A. Pérez, C. Rueda, and F. Valencia. Timed Concurrent Constraint Programming for Analysing Biological Systems. *Electr. Notes Theor. Comp. Sci.*, 171(2):117–137, 2007.

[11] H. Hansson and B. Jonsson. A Logic for Reasoning about Time and Reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[12] O. M. Herescu and C. Palamidessi. Probabilistic asynchronous π-calculus. In *Proc. of FOSSACS 2000*, volume 1784 of *LNCS*, pages 146–160. Springer, 2000.

[13] J. Hillston. Process algebras for quantitative analysis. In *Proc. of 20th LICS*. IEEE Computer Society Press, 2005.

[14] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. 12th TACAS*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[15] M. Nielsen, C. Palamidessi, and F. Valencia. Temporal concurrent constraint programming: Denotation, logic and applications. *Nordic Journal of Computing*, 9(2):145–188, 2002.

[16] C. Olarte, C. Rueda, and F. D. Valencia. Concurrent Constraint Programming: Calculi, Languages and Emerging Applications. *Newsletter of the ALP*, 2008. To appear. Preliminary version in `http://www.lix.polytechnique.fr/~colarte/ccp-alp.pdf`.

[17] C. Olarte and F. D. Valencia. Universal concurrent constraint programing: symbolic semantics and applications to security. In *Proc. of SAC*, pages 145–150. ACM, 2008.

[18] Oxford University. PRISM Model Checker, 2008. See `http://www.prismmodelchecker.org`.

[19] C. Palamidessi. Probabilistic and nondeterministic aspects of anonymity. *Electr. Notes Theor. Comput. Sci.*, 155:33–42, 2006.

[20] A. D. Pierro and H. Wiklicky. An operational semantics for probabilistic concurrent constraint programming. In *Proc. of ICCL*, pages 174–183. IEEE, 1998.

[21] C. Rueda, G. Assayag, and S. Dubnov. A Concurrent Constraints Factor Oracle Model for Music Improvisation. In *Proc. of CLEI'06*, 2006.

[22] C. Rueda and F. Valencia. On validity in modelization of musical problems by ccp. *Soft Computing*, 8(9):641–648, 2004.

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

[24] V. Saraswat, R. Jagadeesan, and V. Gupta. Foundations of timed concurrent constraint programming. In *Proc. of LICS*. IEEE, 1994.

[25] V. Saraswat, M. Rinard, and P. Panangaden. The semantic foundations of concurrent constraint programming. In *POPL '91*, pages 333–352, 1991.

[26] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, 1995.

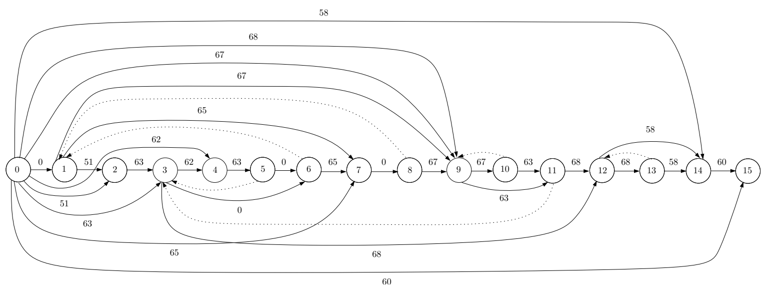


Figure 3: Factor Oracle automaton for the sequence s = (0,51,63,62,63,0,65,0,67,67,63,68,68,58,60).