Modeling and Verification of Real-Time Systems:
Formalisms and Software Tools
Edited by Stephan Merz & Nicolas Navet

Chapter 9

Verification of Probabilistic Systems Methods and Tools

9.1. Introduction

Historically, functional verification and performance evaluation have been two distinct stages in the development of applications. Each one had its own models and methods. For 15 years, numerous works covered both areas. These works are now referred to as *probabilistic verification* or, more accurately, by *verification of probabilistic systems*.

This direction of research is prompted by the new needs of the modelers. They wish, for instance, to compute the probability that a property expressed as some logical formula is satisfied. They also wish to analyze a system including both nondeterministic and probabilistic features. The goal of this chapter is to introduce three important topics related to this research:

– the definition of high level stochastic models;

– the verification of Markov chains (MC);

– the verification of Markov decision processes (MDP).

We will always follow the same organization for these topics:

– the detailed presentation of one approach;

– an overview of the other approaches;

– the description of an analysis tool related to this topic.

Chapter written by Serge HADDAD and Patrice MOREAUX.

The first part of this chapter consists of preliminary notions related to stochastic processes and Markov chains. The second part is devoted to high level formalisms. The third part covers the verification of Markov chains and the chapter ends with a discussion of the verification methods for Markov decision processes.

9.2. Performance evaluation of Markovian models

9.2.1. *A stochastic model of discrete event systems*

We assume that the reader is familiar with the basics of probability theory. For more details, see the following books [FEL 68, FEL 71, TRI 82].

In this chapter, we use the following notations:

– $\Pr(E)$ denotes the probability of an event E and $\Pr(A \mid B)$ the probability of A given B;

– the adverb "almost", in expressions such as "almost everywhere" or "almost surely", means for a set of probability 1;

– $\mathbb{R}$ (respectively $\mathbb{R}^+, \mathbb{R}^{+*}$) denotes the set of real numbers (respectively non negative real numbers, positive real numbers). If x is a real number then $\lfloor x \rfloor$ denotes the greatest integer less than or equal to x;

– if $E \subseteq \mathbb{R}$ then $\mathrm{Inf}(E)$ (respectively $\mathrm{Sup}(E)$) denotes the greatest lower bound (respectively least upper bound) of E.

An execution of a discrete event system (DES) is characterized by a sequence (*a priori* infinite) of events $\{e_1, e_2, \ldots\}$ separated by time delays. Only the occurrence of an event changes the state of the system.

More formally, the stochastic behavior of a DES is defined by two families of random variables:

– $X_0, \ldots, X_n, \ldots$ ranging over the (discrete) space of states, denoted S. In the sequel, unless explicitly mentioned, we assume that this space is finite. X_0 represents the initial state of the system and X_n $(n > 0)$ the current state after the occurrence of the n^{th} event. The occurrence of an event does not necessarily change the state of the system, hence X_{n+1} may be equal to X_n;

– $T_0, \ldots, T_n, \ldots$ ranging over $\mathbb{R}^+$ where T_0 represents the delay before the first event and T_n $(n > 0)$ represents the time elapsing between the n^{th} and the $(n+1)^{th}$ event. Observe that these delays may be zero (e.g., a sequence of instructions considered as instantaneous compared to database transactions including I/O operations).

When initial distribution X_0 is concentrated in a single state s, we say that the process starts in s (i.e., $\Pr(X_0 = s) = 1$).

A priori, there is no restriction on these families of random variables. However, for the categories of processes that we study, a DES cannot execute an infinite number of actions in a finite time. Otherwise stated:

$$\sum_{n=0}^{\infty} T_n = \infty \text{ almost surely.} \tag{9.1}$$

This property allows us to define the state of the system at any instant. Let $N(\tau)$ be the random variable defined by:

$$N(\tau) =_{def} \text{Inf}\left(\left\{ n \mid \sum_{k=0}^{n} T_k > \tau \right\}\right)$$

Due to (9.1), $N(\tau)$ is defined almost everywhere. As can be seen in Figure 9.1, $N(\tau)$ presents jumps strictly greater than 1. The state $Y(\tau)$ of the system at instant τ, is now $X_{N(\tau)}$. Observe that $Y(\tau)$ is not equivalent to the stochastic process, but that it allows, in most cases, to proceed to standard analyses. The scheme in Figure 9.1 presents a possible *realization* of the process and illustrates the meaning of the random variables introduced above. In this example, the process is initially in state s_4 and remains in this state until τ_0 where it visits s_6. At time $\tau_0 + \tau_1$, the system successively visits in zero time, states s_3 and s_{12} before reaching s_7 where it remains some non-zero amount of time. Observing $Y(\tau)$ in continuous time hides the vanishing states s_3 and s_{12} of the process.

The performance evaluation of a DES leads to two kinds of analysis:

– the study of the transient behavior, i.e. the computation of measures depending on the time elapsed since the initial state. This study aims to analyze the initialization stage of a system and the terminating systems. For instance, dependability and reliability [LAP 95, MEY 80, TRI 92] require transient analysis;

– the study of steady-state behavior of the system. In numerous applications, the modeler is interested by the behavior of the system once it is stabilized.

Obviously, this requires that such a steady-state behavior exists. This condition can be expressed, denoting $\boldsymbol{\pi}(\tau)$ the distribution of $Y(\tau)$, by:

$$\lim_{\tau \to \infty} \boldsymbol{\pi}(\tau) = \boldsymbol{\pi} \tag{9.2}$$

where $\boldsymbol{\pi}$ is also a distribution called the *steady-state distribution*.

Transient and steady-state distributions are often only intermediate values useful to compute *performance indices*. For instance, the steady-state probability that a server is available, the probability that at time τ a connection is established or the mean number of clients waiting for a service are such indices.

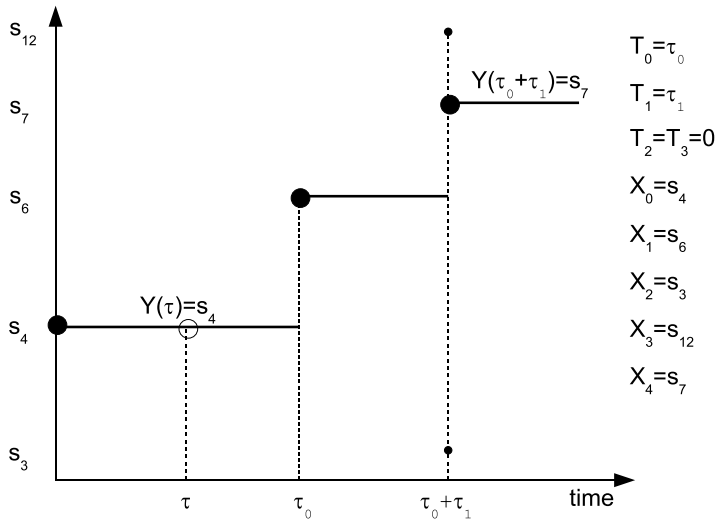

Figure 9.1. *A realization (trajectory) of the stochastic process*

In order to reason in a generic way about DES we assume in the following that we are given a set of functions defined on the set of states and ranging over $\mathbb{R}$. Such a function f may be interpreted as a performance index and, given a distribution $\boldsymbol{\pi}$, quantity $\sum_{s \in S} \boldsymbol{\pi}(s) \cdot f(s)$ represents the measure of this index.

When the index is a function ranging over $\{0, 1\}$, it can be viewed as an atomic proposition satisfied in a state if the function is equal to 1. In the following, we note $\mathcal{P}$, the set of atomic propositions and $s \vDash \phi$, and the fact that s satisfies ϕ, with s a state and ϕ an atomic proposition. In this case, given a distribution $\boldsymbol{\pi}$, quantity $\sum_{s \vDash \phi} \boldsymbol{\pi}(s)$ represents the measure of this index.

9.2.2. *Discrete-time Markov chains*

9.2.2.1. *Presentation*

A discrete-time Markov chain (DTMC) presents the following characteristics:

– the delay between successive instants T_n is the constant value 1;

– the successor of the current state depends only on this state and the transition probabilities are time-invariant[1]:

$$\begin{aligned}&\Pr(X_{n+1} = s_j \mid X_0 = s_{i_0}, \ldots, X_n = s_i)\\ &\quad = \Pr(X_{n+1} = s_j \mid X_n = s_i) = p_{ij} =_{def} \mathbf{P}[i, j].\end{aligned}$$

1. Hence the name *homogenous* chain used in studies about more general definitions of Markov chains.

In the following sections, we use both notations for state transitions.

9.2.2.2. *Transient and steady-state behaviors of DTMC*

In this section we recall classical results while providing intuitive justifications which are not mathematical proofs.

The analysis of the transient behavior does not present any difficulty. State changes occur at time $\{1, 2, \ldots\}$. Given an initial distribution $\boldsymbol{\pi}_0$ and a transition matrix $\mathbf{P}$, then $\boldsymbol{\pi}_n$ the distribution of X_n (i.e. the state of the chain at time n) is given by formula $\boldsymbol{\pi}_n = \boldsymbol{\pi}_0 \cdot \mathbf{P}^n$ which is obtained with the help of an elementary recurrence.

The analysis of the asymptotic behavior of a DTMC (in the case of a countable or finite set of states) leads to the following classification of states:

– a state s is *transient* if the probability that it occurs again once it has occurred is strictly less than 1. Consequently, its occurrence probability $\Pr(X_n = s)$ goes to 0 when n goes to ∞. A state is called *recurrent* if it is not transient;

– a state is *null recurrent* if the mean time between two successive occurrences of this state is infinite. Intuitively, once reached, the mean delay between occurrences of this state will go to ∞ as the number of occurrences goes to ∞ and consequently, once again, the occurrence probability will go to 0. This intuitive reasoning is mathematically sound;

– a state is *non-null recurrent* if the mean delay between two successive occurrences of this state is finite. If a stationary distribution exists then it is concentrated on the non-null recurrent states.

We discuss this analysis when the state space is finite. Let us consider the graph defined as follows:

– the set of vertices is the set of states of the Markov chain;

– there is an edge from s_i to s_j if $p_{ij} > 0$.

Let us study the strongly connected components (SCC) of this graph. If an SCC has an outgoing edge then necessarily the states of this SCC are transient. Conversely, all the states of a *terminal* SCC (i.e. without an outgoing edge) are non-null recurrent. In the particular case where a terminal SCC is reduced to a state s (i.e. $\mathbf{P}[s, s] = 1$), we say that s is an *absorbing* state.

When the graph is strongly connected, we say that the chain is *irreducible*. In the general case, every terminal SCC is an irreducible subchain.

Let us study the existence of a steady-state distribution when the chain is irreducible. First notice that it may not exist. For instance, a chain with two states s_0 and s_1, an initial distribution concentrated in a state and where $p_{0,1} = p_{1,0} = 1$, alternates

between the two states and thus does not converge to a steady-state distribution. By generalization, an irreducible chain is *periodic* with period $k > 1$ if we can partition the states in subsets $S_0, S_1, \ldots, S_{k-1}$ such that from states of S_i the chain only reaches, in one step, states of $S_{(i+1) \bmod k}$.

It turns out that an irreducible and aperiodic chain (called *ergodic*) yields a steady-state distribution and that it is *independent from the initial distribution*. Computing this distribution is relatively easy. Indeed, we have $\boldsymbol{\pi}_{n+1} = \boldsymbol{\pi}_n \cdot \mathbf{P}$. With the limit (which is mathematically sound), we obtain $\boldsymbol{\pi} = \boldsymbol{\pi} \cdot \mathbf{P}$. Moreover, $\boldsymbol{\pi}$ is the single distribution of:

$$\mathbf{X} = \mathbf{X} \cdot \mathbf{P} \tag{9.3}$$

Let us note that an initial distribution, the solution of this equation, is *invariant*: whatever an observation time, the current distribution is identical to the initial distribution. In order to solve equation (9.3), we can proceed to a direct computation by substituting the normalization equation ($\mathbf{X} \cdot \mathbf{1}^T = 1$ where $\mathbf{1}^T$ denotes the column vector with all 1s) for any other equation.

However, iterative computations are more interesting if the state space is huge. The simplest one consists of iterating $\mathbf{X} \leftarrow \mathbf{X} \cdot \mathbf{P}$ [STE 94].

Let us tackle the general case assuming only that the terminal SCC (denoted $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$) are aperiodic with steady-state distributions $\{\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_k\}$. In this case, the chain also generates a stationary distribution (which here depends on the initial distribution). This distribution is given by formula $\boldsymbol{\pi} = \sum_{i=1}^{k} \Pr(\text{to reach } \mathcal{C}_i) \cdot \boldsymbol{\pi}_i$. Thus, it remains to compute the probability of reaching a terminal SCC. We evaluate this quantity starting from a fixed state and then condition it following the initial distribution: $\Pr(\text{to reach } \mathcal{C}_i) = \sum_{s \in S} \boldsymbol{\pi}_0(s) \cdot \boldsymbol{\pi}'_{\mathcal{C}_i}(s)$ where $\boldsymbol{\pi}'_{\mathcal{C}_i}(s) = \Pr(\text{to reach } \mathcal{C}_i \mid X_0 = s)$. Let $\mathbf{P}_{T,T}$ be the transition submatrix of the chain restricted to the transient states and $\mathbf{P}_{T,i}$ the transition submatrix from transient states to states of $\mathcal{C}_i$, then $\boldsymbol{\pi}'_{\mathcal{C}_i} = (\sum_{n \geqslant 0} (\mathbf{P}_{T,T})^n) \cdot \mathbf{P}_{T,i} \cdot \mathbf{1}^T = (\mathbf{I} - \mathbf{P}_{T,T})^{-1} \cdot \mathbf{P}_{T,i} \cdot \mathbf{1}^T$. The first equality is obtained by conditioning the reachability of $\mathcal{C}_i$ by the possible length of the associated path whereas the second can be straightforwardly checked.

9.2.3. *Continuous-time Markov chains*

9.2.3.1. *Presentation*

A continuous-time Markov chain (CTMC) presents the following characteristics:

– the delay between successive instants T_n is a random variable whose distribution is a negative exponential with a rate only depending on state X_n. Otherwise stated:

$$\begin{aligned} &\Pr(T_n \leqslant \tau \mid X_0 = s_{i_0}, \ldots, X_n = s_i, T_0 \leqslant \tau_0, \ldots, T_{n-1} \leqslant \tau_{n-1}) \\ &\quad = \Pr(T_n \leqslant \tau \mid X_n = s_i) = 1 - e^{-\lambda_i \cdot \tau}; \end{aligned}$$

– the successor state of the current state only depends on this state and transition probabilities are time-invariant[2]:

$$\begin{aligned}&\Pr(X_{n+1} = s_j \mid X_0 = s_{i_0}, \ldots, X_n = s_i, T_0 \leqslant \tau_0, \ldots, T_{n-1} \leqslant \tau_{n-1})\\ &\quad = \Pr(X_{n+1} = s_j \mid X_n = s_i) = p_{ij} =_{def} \mathbf{P}[i,j].\end{aligned}$$

The discrete chain defined by matrix $\mathbf{P}$ is called *embedded chain*. It "observes" the state changes of the CTMC without taking into account time elapsing. A state of the CTMC is *absorbing* if it is absorbing with respect to the embedded DTMC.

9.2.3.2. *Transient and steady-state behaviors of CTMC*

In a CTMC, due to the memoryless characteristic of the exponential law, at any time the evolution of DES only depends on its current state.

More precisely, the process is characterized by its initial distribution $\boldsymbol{\pi}(0)$, matrix $\mathbf{P}$ and the family of λ_is. Let us call $\boldsymbol{\pi}(\tau)$ the distribution of $Y(\tau)$ and $\pi_k(\tau) = \boldsymbol{\pi}(t)(s_k)$. If δ is small then the probability that more than one event occurs between τ and $\tau + \delta$ is negligible and the probability that one event occurs and triggers a state change from k to k' is approximatively equal to $\lambda_k \cdot \delta \cdot p_{kk'}$ (by definition of the exponential law).

$$\pi_k(\tau + \delta) \approx \pi_k(\tau) \cdot (1 - \lambda_k \cdot \delta) + \sum_{k' \neq k} \pi_{k'}(\tau) \cdot \lambda_{k'} \cdot \delta \cdot p_{k'k}$$

Thus:

$$\frac{\pi_k(\tau + \delta) - \pi_k(\tau)}{\delta} \approx \pi_k(\tau) \cdot (-\lambda_k) + \sum_{k' \neq k} \pi_{k'}(\tau) \cdot \lambda_{k'} \cdot p_{k'k}$$

And finally:

$$\frac{d\pi_k}{d\tau} = \pi_k(\tau) \cdot (-\lambda_k) + \sum_{k' \neq k} \pi_{k'}(\tau) \cdot \lambda_{k'} \cdot p_{k'k}.$$

Let us define matrix $\mathbf{Q}$ by: $q_{kk'} = \lambda_k \cdot p_{kk'}$ for $k \neq k'$ and $q_{kk} = -\lambda_k (= -\sum_{k' \neq k} q_{kk'})$. Then the previous equation can be rewritten as:

$$\frac{d\boldsymbol{\pi}}{d\tau} = \boldsymbol{\pi} \cdot \mathbf{Q}. \tag{9.4}$$

2. Here again, we say that the chain is *homogenous*.

Matrix $\mathbf{Q}$ is called the *infinitesimal generator* of the CTMC. From (9.4), this generator entirely specifies the chain evolution.

If this equation establishes the memoryless feature of a CTMC, it does not provide a practical means to compute the transient behavior of the chain. For this, we describe a second CTMC equivalent to the first one from a probabilistic point of view (a technique introduced in [JEN 53] and known as uniformization). Let us choose a value $\mu \geqslant \mathrm{Sup}(\{\lambda_i\})$. Whatever the current state, the delay before the next event follows an exponential law with (uniform) parameter μ. The state change is then triggered by transition matrix $\mathbf{P}^\mu$ defined by $\forall i \neq j, \mathbf{P}^\mu[s_i, s_j] = (\mu)^{-1} \cdot \lambda_i \cdot \mathbf{P}[s_i, s_j]$. The DTMC associated with matrix $\mathbf{P}^\mu$ is called the *uniformized chain* (for μ) of the original CTMC. The (straightforward) computation of the infinitesimal generator of the second CTMC shows that it is equal to that of the first chain. So this is the same stochastic process (forgetting the vanishing states). The transient distribution $\boldsymbol{\pi}(\tau)$ is obtained as follows. We compute the probability of being in s at time τ knowing that there have been n state changes during $[0, \tau]$. This probability is given by the uniformized chain and more precisely by $\boldsymbol{\pi}(0) \cdot (\mathbf{P}^\mu)^n$. Afterwards, we "uncondition" this probability by computing the probability of n state changes knowing that the delay between two changes follows the exponential law. This probability is given by $e^{-\mu \cdot \tau} \cdot (\mu \cdot \tau)^n / n!$. Thus:

$$\boldsymbol{\pi}(\tau) = \boldsymbol{\pi}(0) \cdot \left(e^{-\mu \cdot \tau} \sum_{n \geqslant 0} \frac{(\mu \cdot \tau)^n (\mathbf{P}^\mu)^n}{n!} \right).$$

In practice, this infinite sum does not raise any difficulties since it converges very quickly and the summation may be stopped as soon as the required precision is greater than $e^{-\mu \cdot \tau} \cdot (\mu \cdot \tau)^n / n!$.

Let us examine the asymptotic behavior of a CTMC. The simplest way to analyze its behavior consists of studying the embedded DTMC of the uniformized chain. As observed during the presentation of this approach, this chain is not unique. Let us select a DTMC obtained with a choice of $\mu > \mathrm{Sup}(\{\lambda_i\})$. In this case, every state s fulfills $\mathbf{P}^\mu[s, s] > 0$ and thus every terminal SCC of this chain is ergodic. This implies that it yields a steady-state distribution. This distribution measures the steady-state probability of the occurrence of a state. However, since the uniform description of the CTMC implies a mean sojourn time identical for every state $(1/\mu)$, it also provides the steady-state distribution of the CTMC.

In the particular (and frequent) case where the embedded chain is ergodic, this distribution is obtained by solving the equation $\mathbf{X} = \mathbf{X} \cdot \mathbf{P}^\mu$. We observe that $\mathbf{P}^\mu = \mathbf{I} + (1/\mu)\mathbf{Q}$. Thus, the distribution is also the unique solution of the equation:

$$\mathbf{X} \cdot \mathbf{Q} = 0 \quad \text{and} \quad \mathbf{X} \cdot \mathbf{1}^T = 1. \tag{9.5}$$

By analogy, we say that the CTMC is ergodic.

9.3. High level stochastic models

The notion of a stochastic process provides a probabilistic framework for DES, and Markov chain are a subclass of processes whose analysis is feasible. However, the modeler wishes to use a specification model at a higher level whose semantics given by a stochastic process then allows a quantitative analysis. It turns out that the stochastic semantics of a high level model raises specific problems. The goal of this section is to show how to identify and solve these problems.

We have chosen to illustrate these semantical features through the model of Petri nets. We first introduce a general model and then a Markovian sub-model in order to show how to obtain the characteristics of the corresponding CTMC. We assume that the reader is aware of the model of ordinary Petri nets. Otherwise, we advise consulting [GIR 03].

9.3.1. *Stochastic Petri nets with general distributions*

The stochastic feature of Petri nets is introduced by considering that a transition has a variable *firing delay* once it is enabled, obtained by sampling some distribution (ranging over $\mathbb{R}^+$) and that firing conflicts are solved by some probabilistic sampling depending on the transition *weights*. Different subclasses of stochastic Petri nets are defined by restricting these kinds of distributions. In the next definition, we do not restrict these distributions in any way.

DEFINITION 9.1.– *A (marked) stochastic Petri net with general laws (GLSPN)* $N = (P, T, Pre, Post, \Phi, w, m_0)$ *is defined by:*

– P*, the finite set of places;*

– T*, with* $P \cap T = \emptyset$*, the finite set of transitions;*

– Pre *(respectively* $Post$*), the backward (respectively forward) incidence matrix from* $P \times T$ *to* $\mathbb{N}$*;*

– Φ*, a function from* T *to the set of distributions ranging over* $\mathbb{R}^+$*, the law of transition delays;*

– w*, a function from* T *to* $\mathbb{R}^{+*}$*, the transition weights;*

– m_0*, a vector of* $\mathbb{N}^P$*, the initial marking.*

The introduction of distributions and weights is not enough to specify the stochastic process associated with the GLSPN. We are going to study the problems related to this specification.

NOTE.– Most of the parameters of this process could depend on the current marking. For sake of readability, we will not consider it in the following discussion.

9.3.1.1. *Choice policy*

Given that some marking has been reached, we need to determine the next transition to fire among the enabled ones. There are two possibilities:

– a probabilistic choice according to a distribution proportional to the weight of enabled transitions. This is a *pre-selection* since the choice takes place before the sampling of the delay;

– conversely, an independent sampling of the delay for every enabled transition followed by the choice of the shortest delay (with a possible re-use of some previous sampling; see below). When there are multiple shortest delays, we perform an additional probabilistic choice according to the weights, called *post-selection*.

The second solution is generally chosen since on the one hand it corresponds to most of the modeling interpretations and on the other hand, with the help of immediate transitions (see section 9.3.4), the pre-selection can be simulated by the post-selection. Observe that, except when the distributions are continuous, the specification of a distribution for a post-selection is required (here by the weight of transitions).

9.3.1.2. *Service policy*

If, given a marking m, a transition t has enabling degree $e = \lfloor \mathrm{Inf}(\{m(p)/Pre[p,t]\}) \rfloor > 1$, we may consider that the marking *provides* e clients to the transition viewed as a server. So when sampling the delay, there are three options depending on the modeled event:

– a single sampling is performed, the transition accepts clients one by one (*single-server* mode);

– e samplings are performed, the transition accepts all the clients (*infinite-server* mode);

– $\mathrm{Inf}(\{e, deg(t)\})$ samplings are performed, the transition accepts no more than $deg(t)$ clients simultaneously; this case generalizes the previous ones (with $deg(t) = 1$ or ∞) (*multiple-server* mode). Here the modeler must specify $deg(t)$ for every transition.

9.3.1.3. *Memory policy*

Once transition t is fired, what is the effect of the sampling of another transition t' for its next firings?

The first possibility consists of forgetting this sampling. If transition t' is still enabled, a new sampling is performed (*resampling memory* mode). With such a semantic, t could model a crash (immediately repaired) of a service specified by t'.

The second possibility consists of storing the sampling decremented by the shortest sampling, only if t' is still enabled (enabling memory PRD (*Preemptive Repeat*

Different) mode). If t' is disabled, this mechanism could model a *time-out* (t') canceled by the firing of t.

The third possibility is identical when the transition is still enabled but keeps the sampling unchanged if t' is disabled. This sampling will be used again when t' becomes enabled (*enabling memory* PRI (*Preemptive Repeat Identical*) mode). Transition t' when disabled could model a job aborted by t in order to be achieved later in the same conditions.

The fourth possibility consists of storing the sampling decremented by the shortest sampling. A transition t' once disabled could model a job suspended by t (*age memory* mode, also called PRS (*Preemptive ReSume*)).

In order to achieve the specification of this policy, we must take into account the case of multiple-server transitions, which requires selecting which samplings should be kept, suspended or forgotten. The simplest solution is a First In First Out (FIFO) policy for samplings. The last sampling is the first one to be suspended or forgotten. Other policies (such as suspend or forget the least advanced "client") may be incompatible with some analysis methods.

It is clear that once these three policies are defined, the stochastic process is determined without ambiguity. Let us now restrict the kinds of delay distributions.

9.3.2. *GLSPN with exponential distributions*

In the original model [FLO 85, MOL 81] (called *stochastic Petri net* or SPN), every transition t has a delay distribution which is a negative exponential with rate $\lambda(t)$ (assuming an enumeration of T, we denote $\lambda_k = \lambda(t_k)$).

Let us examine the stochastic process generated by such a net with *single-server* mode. Let m be a marking, $t_1, \ldots, t_k$ the enabled transitions from m. We check that:

– the sojourn time in m is an exponential law with rate $\lambda_1 + \cdots + \lambda_k$;

– the probability of firing t_i before the other transitions is equal to $\frac{\lambda_i}{\sum \lambda_j}$ and it does not depend on the elapsed sojourn time;

– the distribution of the remaining firing delay of t_i knowing that t_j is fired is identical to its initial distribution (memoryless law).

Otherwise stated, the current marking fully determines the future behavior of the stochastic process. Thus this process is a continuous-time Markov chain, "isomorphic" to the reachability graph of the net, whose parameters are deduced from the states (i.e. the reachable markings). This reasoning can be generalized to the other modes. Assuming that the graph is finite, the transient and steady-state analyses of this model are performed as described in section 9.2.3.

9.3.3. *Performance indices of SPN*

Since the state of a Petri net is a marking, the functions associated with indices are expressed by numerical expressions where variables are the possible place markings (x_p representing the marking of place p). For instance, assume there is a place cli counting the number of clients and a place *off* witnessing the fact that the server is unavailable. The expression x_{cli} provides the number of clients whereas $x_{cli} \geqslant 1 \wedge x_{off} = 1$ means that at least one client is waiting for a service during the unavailability of the server.

Observe that, using some ad hoc reasoning, it is possible to compute indices related to transitions. For instance, assume that t represents the arrival of a client and that we want to compute the probability that a client arrives when the server is unavailable. First we identify markings where t is enabled. For any such marking m, we compute the probability that t is the next transition to be fired. In SPNs, this probability that we denote $\pi_{fire}(m,t)$ is obtained as the ratio between its rate and the sum of the rates of enabled transitions from m. Let $\boldsymbol{\pi}$ be the state distribution for which the index has to be measured, then the searched value is:

$$\frac{\sum_{m \xrightarrow{t} m' \wedge m'(off)=1} \boldsymbol{\pi}(m) \cdot \pi_{fire}(m,t)}{\sum_{m \xrightarrow{t} m'} \boldsymbol{\pi}(m) \cdot \pi_{fire}(m,t)}.$$

9.3.4. *Overview of models and methods in performance evaluation*

The domain of performance evaluation is extremely vast due to the long history of telecommunications. Thus, we limit ourselves to skipping through this area, referring to books for the interested reader.

The first evaluation models were the queues and then the queuing networks [KLE 75, KLE 76]. This model mainly focuses on the kinds of laws, the type and the number of clients, the service policies and the routing between queues. However, although appropriate for studying telecommunication networks, it misses generic mechanisms in order to model systems (such as operating systems) which include synchronization between components. We can add ad hoc mechanisms but it is safer to add probabilistic features to functional models of concurrent systems.

Stochastic Petri nets have been the support of numerous modelings with ordinary Petri nets [AJM 95]. Their elementary firing rule makes it possible to easily transform them into a stochastic model as we have done in the previous section. Furthermore, high-level Petri nets which provide support for data structure and parametrization of actions have also be enlarged with a stochastic semantics [CHI 93b].

Process algebra integrate compositional features and thus are appropriate for a hierarchical design. So, they have also been enlarged with a stochastic semantics [HIL 96]. Contrary to Petri nets, this semantic raises subtle problems like, for instance, the quantitative specification of action synchronization.

The evaluation methods are generally classified with respect to the complexity of computations. We follow this order restricting the references to the ones related to Petri nets for sake of conciseness.

When the structure of a model is really simple, it is possible to obtain a formula expressing the steady-state distribution with respect to the numerical parameters. In the case of a formula obtained by sub-formulae associated with the system components, we call such formula a product form. In the framework of Petri nets, a product form is difficult to obtain due to the synchronization required for firing a transition. However, for subclasses of Petri nets such formulae have been established [HEN 90, HAD 05].

In the general case, it is necessary to generate the CTMC associated with the SPN and to compute its steady-state distribution. When the size of the chain is too large, an analysis of the structure of the net makes it possible to compute bounds [CHI 93a] or to design approximate methods [CAM 94]. In the case of high-level nets, the aggregation techniques *a priori* generate a smaller CTMC equivalent to the CTMC of the net [CHI 93b]. When the Petri net is unbounded (i.e. when the place markings may be arbitrarily large), the associated CTMC is infinite. However, if a single place is unbounded, it is still possible to obtain the steady-state distribution [HAV 95].

If exponential laws are suitable to model events whose time distribution is unknown, some operations have a duration belonging to some time interval or even close to be constant. In such cases, the choice of an exponential law leads to loose approximations. Thus, a study of nets including transitions with deterministic laws (also called Dirac laws) has been undertaken [AJM 87, LIN 98]. Numerous alternative approaches have then been proposed depending on the kind of laws and the occurrence of the corresponding transitions [DON 98, GER 99, LIN 98, LIN 99].

9.3.5. *The GreatSPN tool*

GreatSPN [AJM 95] is one of the most prominent tools for the qualitative and quantitative analysis of Petri nets, developed since the beginning of the 1980s by the performance evaluation team of Torino University. First analyzing SPNs, it has gradually integrated the semantical extensions related to the transition distributions. It is also the only tool to cope with the model of the stochastic well-formed Petri net (SWN), a high-level model which takes advantage of behavioral symmetries in order to obtain a lumped CTMC directly from the definition of the net. GreatSPN is available from the authors, free for academic use. The software includes a graphical interface for the

definition of the net, which also handles non-graphical properties (like delays) and triggers the computations. Most of the results are presented in the graphical interface.

9.3.5.1. *Supported models*

GreatSPN manages ordinary Petri nets whose transitions have negative exponential distributions (standard SPN), but also phase type distribution (like ERLANG), deterministic distribution (i.e. constant) and the null Dirac distribution (immediate transitions). GreatSPN also manages stochastic well-formed Petri nets, the most widely used high-level stochastic Petri net model. Both models are analyzed using the graphical interface.

9.3.5.2. *Qualitative analysis of Petri nets*

The tool integrates most of the standard analysis methods for Petri nets: boundedness checking, linear invariant computations, trap and deadlock computations, etc. When the net is bounded, GreatSPN computes and saves on disk its reachability graph. For an SWN, it computes a symbolic reachability graph which can be transformed into a lumped Markov chain.

9.3.5.3. *Performance analysis of stochastic Petri nets*

GreatSPN computes the transient and steady-state distributions of bounded nets. The user can also specify complex performance indices that the tool computes on demand. The software also provides distributions and performance indices for SWN at the lumped level and if necessary at the ordinary level.

Despite numerous improvements for the state representation (i.e. the markings), the size of the reachability graph may forbid the exact resolution of very large models. Thus, GreatSPN includes a stochastic simulator in order to cope with such situations.

9.3.5.4. *Software architecture*

The implementation of GreatSPN is based on a modular structure which follows the analysis process for Petri nets (definition, qualitative analysis, performance evaluation). Every step is realized by programs written in C, whose executable code can also be triggered by commands. This makes it possible to write scripts in order to combine the different algorithms implemented in the tool. The reader can refer to the site `www.unito.it/~GreatSPN` for more details.

9.4. Probabilistic verification of Markov chains

The detailed discussion refers to CTMC.

9.4.1. *Limits of standard performance indices*

Performance indices defined above give valuable information to the system designer. However, they do not express all significant measures. Let us illustrate this point with the help of a service availability. Some properties related to this concept are as follows:

– *instantaneous* availability guarantee in transient mode. That is, the probability, at a given time τ, of the service availability;

– instantaneous availability guarantee in steady-state. That is, the probability, at any time, of the service availability in steady-state;

– sustained availability guarantee in transient mode. That is, the probability that the system is permanently available between times τ and τ';

– sustained availability guarantee in steady-state. That is, the probability that the service is permanently available between two instants in steady-state. Since the process is in steady-state, this index depends only on the duration between these two instants;

– availability and response time guarantee in steady-state. That is, the probability that, after a request, the service stays on until the response and that the response time is lower than a given upper bound.

Although the first two properties may be easily deduced from the transient and steady-state distributions, this is not the case for the other properties. We could figure out an ad hoc algorithm for each of these. It is however more judicious to introduce a new logic to express complex performance indices and to design a general evaluation algorithm for this logic.

9.4.2. *A temporal logic for Markov chains*

The continuous stochastic logic (CSL) is an adaptation of the computation tree logic (CTL) [EME 80] to continuous Markov chains. It expresses formulae evaluated on states with the following syntax. Here we mainly refer to the approach of [BAI 03a].

DEFINITION 9.2.– *A CLS formula is inductively defined as:*

– if $\phi \in \mathcal{P}$, then ϕ is a CSL formula;

– if ϕ and ψ are CSL formulae, then $\neg\phi$ and $\phi \wedge \psi$ are CSL formulae;

– if ϕ is a CSL formula, $a \in [0,1]$ is a real number, $\bowtie \in \{<, \leqslant, >, \geqslant\}$, then $S_{\bowtie a}\phi$ is a CSL formula;

– if ϕ and ψ are CSL formulae, $a \in [0,1]$ is a real number, $\bowtie \in \{<, \leqslant, >, \geqslant\}$ and I is an interval of $\mathbb{R}_{\geqslant 0}$, then $P_{\bowtie a}\mathcal{X}^I\phi$ and $P_{\bowtie a}\,\phi\mathcal{U}^I\psi$ are CSL formulae.

Only the two last items require some explanation. Formula $S_{\bowtie a}\phi$ is satisfied by a state s of the chain if, for the process started in s, the stationary cumulated probability (say p) of the states satisfying ϕ verifies $p \bowtie a$. The value of this formula is well defined since a finite CTMC has a stationary distribution. Let us note that this value does not depend on state s if the chain is ergodic.

A sample of the stochastic process satisfies $\mathcal{X}^I\phi$ if the first state change occurs in the interval I and if the reached state verifies ϕ. State s satisfies $P_{\bowtie a}\mathcal{X}^I\phi$ if the probability (say p) that a sample of the process started in s satisfies the given condition fulfills $p \bowtie a$.

A sample of the stochastic process satisfies $\phi\mathcal{U}^I\psi$ if there is a time $\tau \in I$ such that ψ is satisfied and at all previous times ϕ is satisfied. State s satisfies $P_{\bowtie a}\phi\mathcal{U}^I\psi$ if the probability (say p) that a sample of the process, started in s satisfies the given condition fulfills $p \bowtie a$.

To exemplify these definitions, let us give the formal expressions of the availability properties expressed above.

– 99% instantaneous availability guarantee in transient mode:

$$P_{\geqslant 0.99} true \mathcal{U}^{[\tau,\tau]} disp$$

where $disp$ is an atomic proposition meaning that the service is available.

– 99% instantaneous availability guarantee in steady-state:

$$S_{\geqslant 0.99} disp$$

– 99% sustained availability guarantee in transient mode:

$$P_{<0.01} true \mathcal{U}^{[\tau,\tau']} \neg disp$$

– 99% sustained availability guarantee in steady-state:

$$S_{<0.01} true \mathcal{U}^{[\tau,\tau']} \neg disp$$

– 99% availability and response time (3 time units) guarantee in steady-state:

$$S_{\geqslant 0.99}(req \Rightarrow P_{\geqslant 0.99}(disp \mathcal{U}^{[0,3]} ack))$$

where req is an atomic proposition meaning a receiving request and ack is an atomic proposition meaning an answer to a request. Let us note that the two 99% occurrences do not have the same meaning. The internal operator occurrence is a requirement on the behavior of the process started in a specific state, while the second one is a global requirement on the states weighed with a stationary distribution. *A priori*, different required values could have been specified.

9.4.3. *Verification algorithms*

Given a CTMC and a CSL formula ϕ, the verification algorithm proceeds by the successive evaluation of the sub-formulae of ϕ, "upwards" in the syntactic tree of the formula ϕ, from leaves to the root, labeling each state with the sub-formulae this state verifies. Thus, every step of the algorithm evaluates a formula viewing the operands of the most external operator as atomic propositions.

This leads us to study each operator.

$\boxed{\phi = \neg\psi}$ The algorithm labels each state with ϕ if it is not labeled with ψ.

$\boxed{\phi = \psi \wedge \chi}$ The algorithm labels each state with ϕ if it is labeled with ψ and χ.

$\boxed{\phi = S_{\bowtie a}\psi}$ The algorithm computes the steady-state distribution of the process started in s (as mentioned in section 9.2.3). Then it sums up the probabilities of the states labeled with ψ and labels s with ϕ if the computed value (say p) verifies $p \bowtie a$. Let us note that, for all states of a sink SCC, only one computation is required. Also, if the CTMC has a unique stationary distribution, then the truth value of the formula is state independent.

$\boxed{\phi = P_{\bowtie a}\mathcal{X}^I\psi}$ Let s be a state. The occurrence of the next transition inside the interval I and the satisfaction of ψ by the reached state are two independent events. Thus, the searched probability is the product of the probabilities of these events. Let us denote $I = [\tau, \tau']$; we assume without loss of generality that intervals are closed. Indeed, since distributions are continuous, taking or not taking interval bounds into account does not matter with regard to the value of the formula. If $\mathbf{Q}$ is the infinitesimal generator of the chain and $\mathbf{P}$ the transition matrix of the embedded Markov chain, then the probability of the first event is $e^{\tau\mathbf{Q}[s,s]} - e^{\tau'\mathbf{Q}[s,s]}$ and the probability of the second event is $\sum_{s' \models \psi} \mathbf{P}[s, s']$.

$\boxed{\phi = P_{\bowtie a}\psi\mathcal{U}^I\chi}$ The evaluation of this formula mainly consists of transient analyses of chains derived from the original chain by elementary transformations. For a chain X, we denote by X^ϕ the chain derived by transforming all states verifying ϕ as absorbing states. To simplify matters, we study the various kinds of intervals.

– $\phi = P_{\bowtie a}\psi\mathcal{U}^{[0,\infty[}\chi$. In this case, the sample of the process must stay in states verifying ψ until reaching a state verifying χ, without time constraint. In other words, we track the behavior of the chain until a state verifying $\neg\psi \vee \chi$. Let us study the chain $X^{\neg\psi\vee\chi}$. If a terminal SCC of this chain holds a state verifying χ, then it is reduced to a single state and the requested probability is 1, otherwise this probability is zero for all states of the SCC since they cannot reach a state verifying χ. Let us call "good" a SCC associated with probability 1. So, the requested probability for the remaining states is the probability of reaching a state of a good SCC. This probability depends only on the embedded chain of $X^{\neg\psi\vee\chi}$ and its computation has been described in section 9.2.2.

– $\phi = P_{\bowtie a}\psi\mathcal{U}^{[0,\tau]}\chi$. In this case, the sample of the process must stay in states verifying ψ until reaching a state verifying χ no later than time τ. In other words, we track the behavior of the chain until reaching a state verifying $\neg\psi \vee \chi$. Thus, the probability to be computed is $\Pr(X^{\neg\psi\vee\chi}(\tau) \vDash \chi \mid X^{\neg\psi\vee\chi}(0) = s)$.

– $\phi = P_{\bowtie a}\psi\mathcal{U}^{[\tau,\tau]}\chi$. In this case, the sample of the process must stay in states verifying ψ during the interval $[0, \tau]$ and moreover these states must verify χ at time τ. We overlook state changes at time τ since its probability is zero. Thus, the probability to be computed is $\Pr(X^{\neg\psi}(\tau) \vDash \psi \wedge \chi \mid X^{\neg\psi}(0) = s)$.

– $\phi = P_{\bowtie a}\psi\mathcal{U}^{[\tau,\infty[}\chi$. In this case, the sample of the process must stay in states verifying ψ during the interval $[0, \tau]$, then it must verify the formula $\psi\mathcal{U}^{[0,\infty[}\chi$ from state s reached at time τ. The requested probability is then $\sum_{s'\vDash\psi} \Pr(X^{\neg\psi}(\tau) = s' \mid X^{\neg\psi}(0) = s) \cdot \boldsymbol{\pi}(s')$ where $\boldsymbol{\pi}(s')$ is computed as in the first case.

– $\phi = P_{\bowtie a}\psi\mathcal{U}^{[\tau,\tau']}\chi$. The same reasoning as above leads to the following formula for the requested probability: $\sum_{s'\vDash\psi} \Pr(X^{\neg\psi}(\tau) = s' \mid X^{\neg\psi}(0) = s) \cdot \Pr(X^{\neg\psi\vee\chi}(\tau' - \tau) \vDash \chi \mid X^{\neg\psi\vee\chi}(0) = s')$.

9.4.4. *Overview of probabilistic verification of Markov chains*

Historically, the verification of discrete-time chains happened before the verification of continuous-time chains. The first attempt for verifying LTL formulae on DTMC [VAR 85] is conceptually easy: translate the formula into a Büchi automaton; then determinize this automaton into a Rabin automaton; build the synchronized product of this automaton with the DTMC, leading to a new DTMC; and apply a variant of the analysis presented in section 9.2. Unfortunately, the complexity of this algorithm is doubly exponential with respect to the size of the formula. In [COU 95], the authors also build a new DTMC by iterative refinement of the initial DTMC analyzing the formula operators. This leads to a simple exponential procedure. Moreover, they show that this is the optimal complexity. A third algorithm [COU 03] also translates the formula into a Büchi automaton. However, the chosen translation algorithm allows us to directly evaluate the probability related to the formula on the synchronized product of the automaton and the DTMC. This method also presents an optimal theoretical complexity and behaves better than the previous method in effective modeling cases.

A standard analysis technique for performance models is to combine "rewards" to states and/or to transitions of a chain and to compute performance indices related to these rewards. In order to extend the scope of probabilistic verification to such models, a new logic, PRCTL, is introduced in [AND 03] together with a formula evaluation algorithm.

The first significant works on CTMC were established in [AZI 96, AZI 00]. They proved that verification of CSL formulae on CTMC is decidable. However, the corresponding algorithm is very complicated since it forbids the approximations we have implicitly used in the computations of the previous section.

In fact, even with the above method, computation may become intractable for large Markov chains. An efficient approach to cope with this problem is to profit from the modularity of the specification. In this context, we try to replace a module with a smaller but equivalent one with respect to the formula to be checked. Then we check the model made up of reduced modules. This approach, initialized in [BAI 03a], was generalized in [BAI 03b] which studies numerous kinds of equivalence.

A completely different approach to reduce complexity of the verification is proposed in [YOU 06]. Assuming we have to check the formula $P_{\leqslant a}\phi$, we can generate random executions and compute the ratio of executions satisfying ϕ; in accordance with classical probability results, this value converges on the requested probability. This method is very efficient when the evaluation of the formula ϕ only requires a time bounded execution.

9.4.5. *The ETMCC tool*

ETMCC (Erlangen-Twente Markov chain model checker) is the first model checking tool (2000) for CSL formulae on CTMC. It is developed by several German universities, originally Erlangen, and the Deutch university of Twente. A limited version is available from the authors. ETMCC allows analysis of CTMC with properties given in CSL extended with several convenient features for the modeler, including rewards. Tool usage is standard: model specification with a specific formalism, definition of the properties with probabilistic temporal logic and results analysis. Even if ETMCC also analyzes discrete-time models, we present functionalities for continuous-time systems for which it was specifically designed. Moreover, the discrete-time case is the subject of common work with the PRISM tool team (see below).

9.4.5.1. *Language of system models*

ETMMC has chosen a very simple solution: models are CTMC, provided by the user as the rate matrix given through textual sparse version. Let us note this allows us to use ETMCC from various higher-level models (process algebras, Petri nets, etc.) as soon we are able to generate their (finite) state spaces. It is enough to translate these descriptions into the textual input format of ETMCC. The list of atomic properties satisfied on states must also be provided through a text file.

9.4.5.2. *Language of properties*

ETMCC allows us to define properties with the CSL logic detailed above. They are given through the graphical user interface. CSL is extended to mean reward computations in transient or steady-state mode, from state or event reward functions; this allows the practitioner to obtain useful indices such as the mean cost of a transaction during a given duration.

9.4.5.3. *Computed results*

The software analyzes the model with respect to required properties and shows results, truth values of CSL formulae and reward indices, through the user interface.

9.4.5.4. *Software architecture*

ETMCC is implemented in Java. The most involved part is the evaluation of bounded time (Next and Until) operators. It makes use of numerical integration solvers or CTMC uniformization solvers. The application also involves graph algorithms especially for computation of maximal SCC. The graph of reachable states is implemented with a sparse representation. This representation is also used for all numerical computations. The reader can visit the ETMCC web site `http://www7.informatik.uni-erlangen.de/etmcc` for a detailed description of the tool.

9.5. Markov decision processes

9.5.1. *Presentation of Markov decision processes*

Let us assume that we have to analyze executions of a set of transactions such that each one can be modeled with a CTMC. Seeking for a model of the global system, we are then faced to the fact that we do not know the scheduler of the transactional system. We could model choices of the scheduler as probabilistic actions and thus, come down to a global CTMC. However, this solution reduces significantly the scope of computed measures. Indeed, such results would be interpreted as performance indices of an "average" scheduler. However, in practice, we are looking for extremal indices such as maximal probability of a transaction abort, for all (an infinite number of) schedulers.

It is thus necessary to use a more expressive formalism than DTMC. More precisely, this formalism must allow us to express probabilistic choices and non-deterministic choices. This naturally leads us to *Markov decision process* (MDP).

DEFINITION 9.3.– *A Markov decision process* $\Sigma = (S, \mathcal{P}, V, next)$ *is defined by:*

– S*, the (finite) set of states;*

– $\mathcal{P}$*, the (finite) set of atomic propositions;*

– V*, the state characteristic function which associates with each state* s*, the subset* $V(s)$ *of* $\mathcal{P}$ *of true propositions in* s*;*

– $next$*, the function which associates with each state* s*, the set* $next(s) = \{\boldsymbol{\pi}_1^s, \ldots, \boldsymbol{\pi}_{k_s}^s\}$ *of distributions with support* S*.*

The fundamental element of this definition is the $next$ function which controls the evolution of the process. In state s, the process non-deterministically chooses a distribution $\boldsymbol{\pi}_i^s \in next(s)$, and then samples this distribution, which defines the next state.

In agreement with this interpretation, we introduce a (immediate) succession relation ρ defined by $\rho(s, s') \Leftrightarrow \exists \boldsymbol{\pi}_i^s,\ \boldsymbol{\pi}_i^s(s') > 0$. An execution path is then a sequence of states such that every pair of consecutive states fulfills this relation.

We wish to put this system in a full probabilistic setting. To do so, we define so called strategies. A strategy t is a function providing for every execution path $s_0, s_1, \ldots, s_n$, $St(s_0, s_1, \ldots, s_n)$, a distribution from $next(s_n)$. For a given strategy and a given initial state, the Markov decision process behaves like a discrete-time stochastic process so that the probability of an event Ev of this process is well defined and will be denoted by $\Pr^{St}(Ev)$.

9.5.2. *A temporal logic for Markov decision processes*

Probabilistic computation tree logic (pCTL), the temporal logic that we will present, is an adaptation of CTL to Markov decision processes. We refer here mainly to the approach of [BIA 95]. Formulae of pCTL are evaluated on states with the following syntax.

DEFINITION 9.4.– *A pCTL formula is inductively defined by:*

– if $\phi \in \mathcal{P}$, then ϕ is a pCTL formula;

– if ϕ and ψ are pCTL formulae, then $\neg\phi$ and $\phi \wedge \psi$ are pCTL formulae;

– if ϕ and ψ are pCTL formulae, $a \in [0, 1]$ is a real number and $\bowtie \in \{<, \leqslant, >, \geqslant\}$, then $A\phi\mathcal{U}\psi$, $E\phi\mathcal{U}\psi$ and $P_{\bowtie a}\phi\mathcal{U}\psi$ are pCTL formulae.

Only the last point requires some explanation. The first two operators do not involve numerical values of the distributions. $A\phi\mathcal{U}\psi$ (respectively $E\phi\mathcal{U}\psi$) is true in state s if and only if every (respectively at least one) execution path starting in s comprises a prefix made of states satisfying ϕ followed by a state satisfying ψ. The last operator involves strategies in the following way: $P_{\bowtie a}\phi\mathcal{U}\psi$ is true in s if *for every strategy*, the probability (say p) for the corresponding stochastic process that an execution path stating in s comprises a prefix made of states satisfying ϕ, followed by a state satisfying ψ, satisfies $p \bowtie a$.

9.5.3. *Verification algorithms*

Given a Markov decision process, and a pCTL formula ϕ, the model checking algorithm proceeds by successive evaluation of sub-formulae of ϕ, "going upwards" the syntactic tree of the formula ϕ, from leaves to the root, and labeling each state with the sub-formulae it satisfies. Thus, each step of the algorithm evaluates a formula viewing the operands of the most external operator as atomic propositions.

This leads us to study each operator.

$\boxed{\phi = \neg\psi}$ The algorithm labels each state with ϕ if it is not labeled with ψ.

$\boxed{\phi = \psi \wedge \chi}$ The algorithm labels each state with ϕ if it is labeled with ψ and χ.

$\boxed{\phi = E\psi\mathcal{U}\chi}$ In a first step, the algorithm labels states labeled with χ. Then, going back up from these states using the precedence relation (ρ^{-1}) it labels states labeled with ψ. It iterates this step starting from the newly labeled states until saturation.

$\boxed{\phi = A\psi\mathcal{U}\chi}$ The algorithm uses a recursive function tagging visited states. When it evaluates a state labeled with χ, it labels it with ϕ and returns true. When it evaluates a state not labeled with χ or ψ it returns false. When it evaluates a state not labeled with χ but labeled with ψ and *not yet visited*, it calls the function for each successor of the state and labels it with ϕ if all calls return true. When it evaluates an *already visited* and not labeled state, it returns false. The reader could check that this procedure returns false if there is a path starting in the state, which comprises a state not labeled with ϕ or χ before every state labeled with χ, or a (infinite) path made of states labeled with ϕ but not with χ. This last case is detected by the presence of a circuit, thanks to tagging states.

$\boxed{\phi = P_{\geqslant a}\psi\mathcal{U}\chi}$ We only study this probabilistic operator since the other ones are similar. The algorithm computes simultaneously for all states the minimal probability (say $\boldsymbol{\pi}_{\min}(s) = \mathrm{Inf}(\{\Pr^{St}(s \vDash \psi\mathcal{U}\chi)\})$ of "good" paths and then compares it with a to find states to be labeled. If a state verifies χ, then whatever the strategy St, $\Pr^{St}(s \vDash \psi\mathcal{U}\chi) = 1$ and so $\boldsymbol{\pi}_{\min}(s) = 1$. Let us call S_{good} this subset of states. From S_{good}, we get the set of states where $\boldsymbol{\pi}_{\min}(s) > 0$. This set, denoted by $S_{>0}$, is first initialized to S_{good} and it is iteratively enlarged with states which have, whatever the strategy, a non-zero probability of reaching it in one step: $S_{>0} \leftarrow S_{>0} \cup \{s \mid \forall \boldsymbol{\pi}^i_s, \exists s' \in S_{>0}, \boldsymbol{\pi}^i_s(s') > 0\}$. This procedure necessary terminates. Let us call $S_{bad} = S \setminus S_{>0}$. We can easily check that $\forall s \in S_{bad}, \boldsymbol{\pi}_{\min}(s) = 0$. It remains to evaluate $S_{int} = S_{>0} \setminus S_{good}$. Since this evaluation is at the heart of the method and its extensions, we explain it in detail below and we prove its correctness.

First claim. Vector $\boldsymbol{\pi}_{\min}$ is a solution of equation (9.6) where vector $\mathbf{x}$ is the unknown.

$$\forall s \in S_{int}, \mathbf{x}(s) = \mathrm{Inf}\left(\left\{ \sum_{s' \in S_{int}} \boldsymbol{\pi}^i_s(s')\mathbf{x}(s') + \sum_{s' \in S_{good}} \boldsymbol{\pi}^i_s(s') \right\}_{1 \leqslant i \leqslant k_s}\right). \quad (9.6)$$

Proof. We state the equality, proving the two inequalities.
($\geqslant$) Let St be a strategy for the process starting in s, then:

$$\Pr^{St}(s \vDash \psi\mathcal{U}\chi) = \sum_{s' \in S_{int}} \boldsymbol{\pi}^{St(s)}_s(s')\Pr^{St_{s'}}(s' \vDash \psi\mathcal{U}\chi) + \sum_{s' \in S_{good}} \boldsymbol{\pi}^{St(s)}_s(s')$$

with $St_{s'}$ the strategy defined by $St_{s'}(s', \ldots, s_n) = St(s, s', \ldots, s_n)$.

So,

$$\Pr^{St}(s \vDash \psi \mathcal{U} \chi) \geqslant \sum_{s' \in S_{int}} \boldsymbol{\pi}_s^{St(s)}(s') \boldsymbol{\pi}_{\min}(s') + \sum_{s' \in S_{good}} \boldsymbol{\pi}_s^{St(s)}(s')$$

$$\geqslant \mathrm{Inf}\left(\left\{ \sum_{s' \in S_{int}} \boldsymbol{\pi}_s^i(s') \boldsymbol{\pi}_{\min}(s') + \sum_{s' \in S_{good}} \boldsymbol{\pi}_s^i(s') \right\}_{1 \leqslant i \leqslant k_s} \right)$$

With this final inequality being true for all St, we get:

$$\forall s \in S_{int}, \boldsymbol{\pi}_{\min}(s) \geqslant \mathrm{Inf}\left(\left\{ \sum_{s' \in S_{int}} \boldsymbol{\pi}_s^i(s') \boldsymbol{\pi}_{\min}(s') + \sum_{s' \in S_{good}} \boldsymbol{\pi}_s^i(s') \right\}_{1 \leqslant i \leqslant k_s} \right).$$

($\leqslant$) Now, let $\epsilon > 0$ given, then, by definition of $\boldsymbol{\pi}_{\min}$, for all s' there is a strategy $St_{s'}$ such that $\Pr^{St_{s'}}(s' \vDash \psi \mathcal{U} \chi) \leqslant \boldsymbol{\pi}_{\min}(s') + \epsilon$. Given a state s, we build a strategy St for the process starting in s in the following way. St chooses the distribution $\boldsymbol{\pi}_s^i$ which minimizes $\sum_{s' \in S_{int}} \boldsymbol{\pi}_s^i(s') \Pr^{St_{s'}}(s \vDash \psi \mathcal{U} \chi) + \sum_{s' \in S_{good}} \boldsymbol{\pi}_s^i(s')$, then applies to the next reached state s' the strategy $St_{s'}$. By construction,

$$\forall i, \Pr^{St}(s \vDash \psi \mathcal{U} \chi) \leqslant \sum_{s' \in S_{int}} \boldsymbol{\pi}_s^i(s') \Pr^{St_{s'}}(s' \vDash \psi \mathcal{U} \chi) + \sum_{s' \in S_{good}} \boldsymbol{\pi}_s^i(s')$$

$$\leqslant \sum_{s' \in S_{int}} \boldsymbol{\pi}_s^i(s') (\boldsymbol{\pi}_{\min}(s') + \epsilon) + \sum_{s' \in S_{good}} \boldsymbol{\pi}_s^i(s')$$

$$\leqslant \epsilon + \sum_{s' \in S_{int}} \boldsymbol{\pi}_s^i(s') \boldsymbol{\pi}_{\min}(s') + \sum_{s' \in S_{good}} \boldsymbol{\pi}_s^i(s')$$

Thus,

$$\boldsymbol{\pi}_{\min}(s) \leqslant \Pr^{St}(s \vDash \psi \mathcal{U} \chi)$$

$$\leqslant \epsilon + \mathrm{Inf}\left(\left\{ \sum_{s' \in S_{int}} \boldsymbol{\pi}_s^i(s') \boldsymbol{\pi}_{\min}(s') + \sum_{s' \in S_{good}} \boldsymbol{\pi}_s^i(s') \right\}_{1 \leqslant i \leqslant k_s} \right)$$

This last equality being true for arbitrary small ϵ, the second equality is established and concludes the proof. □

Second claim. Vector $\boldsymbol{\pi}_{\min}$ is the *unique* solution of (9.6).

Proof. To establish uniqueness, we study *Markovian* strategies, that is to say strategies for which the chosen distribution depends only on the last state of the execution. Let us denote $St(s)$ the chosen distribution when this state is s. Let us also note that

the behavior of the process is then a DTMC. The equation satisfied by a Markovian strategy is:

$$\forall s \in S_{int}, \quad \Pr^{St}(s \vDash \psi \mathcal{U} \chi) = \sum_{s' \in S_{int}} \boldsymbol{\pi}_s^{St(s)}(s') \Pr^{St}(s' \vDash \psi \mathcal{U} \chi) + \sum_{s' \in S_{good}} \boldsymbol{\pi}_s^{St(s)}(s') \tag{9.7}$$

To simplify notations, $\mathbf{x}(s)$ will stand for $\Pr^{St}(s \vDash \psi \mathcal{U} \chi)$, $\mathbf{A}[s, s']$ will stand for $\boldsymbol{\pi}_s^{St(s)}(s')$ and $\mathbf{b}(s)$ will stand for $\sum_{s' \in S_{good}} \boldsymbol{\pi}_s^{St(s)}(s')$. Equation (9.7) is then rewritten in vector notation $\mathbf{x} = \mathbf{A} \cdot \mathbf{x} + \mathbf{b}$. Quantity $\mathbf{A}^n[s, s']$ is the probability of being in s' starting from s after n steps without leaving S_{int}. From the definition of S_{int}, for every strategy, the probability of staying forever in S_{int} is zero, which is expressed in DTMC context by the convergence of the series $\sum_{n \geqslant 0} \mathbf{A}^n$ (see section 9.2.2). Replacing (n times) $\mathbf{x}$ with its expression in the right-hand side of the equality, we get $\mathbf{x} = \sum_{i \leqslant n} \mathbf{A}^i \mathbf{b} + \mathbf{A}^{n+1} \mathbf{x}$. Taking the limit, $\mathbf{x} = \sum_{n \geqslant 0} \mathbf{A}^n \mathbf{b}$, which means that (9.7) has a unique solution.

Now, let $\mathbf{x}$ be a solution of (9.6). Let us denote by St the Markovian strategy which chooses for a state s, the distribution $\boldsymbol{\pi}_s^i$ for which the minimum is reached with the solution $\mathbf{x}$. Then $\mathbf{x}$ satisfies (9.7) corresponding to this strategy. Thus, $\mathbf{x}(s) = \Pr^{St}(s \vDash \psi \mathcal{U} \chi)$. We deduce that $\forall s, \mathbf{x}(s) \geqslant \boldsymbol{\pi}_{\min}(s)$. Now, let St' be the Markovian strategy leading to $\boldsymbol{\pi}_{\min}$. We note that, for every $s \in S_{int}$:

$$\begin{aligned}
&\sum_{s' \in S_{int}} \boldsymbol{\pi}_s^{St'(s)}(s')(\mathbf{x}(s') - \boldsymbol{\pi}_{\min}(s')) \\
&\quad = \left(\sum_{s' \in S_{int}} \boldsymbol{\pi}_s^{St'(s)}(s') \mathbf{x}(s') + \sum_{s' \in S_{good}} \boldsymbol{\pi}_s^{St'(s)}(s') \right) \\
&\qquad - \left(\sum_{s' \in S_{int}} \boldsymbol{\pi}_s^{St'(s)}(s') \boldsymbol{\pi}_{\min}(s') + \sum_{s' \in S_{good}} \boldsymbol{\pi}_s^{St'(s)}(s') \right) \\
&\quad \geqslant \mathbf{x}(s) - \boldsymbol{\pi}_{\min}(s)
\end{aligned}$$

Rewritten with previous vector notations, this is given by $\mathbf{A}(\mathbf{x} - \boldsymbol{\pi}_{\min}) \geqslant \mathbf{x} - \boldsymbol{\pi}_{\min} \geqslant \mathbf{0}$. By iteration, we get $\mathbf{A}^n(\mathbf{x} - \boldsymbol{\pi}_{\min}) \geqslant \mathbf{x} - \boldsymbol{\pi}_{\min} \geqslant \mathbf{0}$. Finally, taking the limit, $\mathbf{x} - \boldsymbol{\pi}_{\min} = \mathbf{0}$ which proves the claim. □

Third claim. Vector $\boldsymbol{\pi}_{\min}$ is the unique solution of the linear programming problem:

$$\text{Maximize} \sum_{s \in S_{int}} \mathbf{x}(s) \text{ under the constraints:}$$

$$\forall s \in S_{int}, \ \forall 1 \leqslant i \leqslant k_s, \ \mathbf{x}(s) \leqslant \sum_{s' \in S_{int}} \boldsymbol{\pi}_s^i(s') \mathbf{x}(s') + \sum_{s' \in S_{good}} \boldsymbol{\pi}_s^i(s').$$

Proof. If $\mathbf{x}$ satisfies the constraints of this problem, then $\mathbf{x}$ satisfies the version of (9.6) where equalities are replaced with inequalities. Let us assume moreover that $\mathbf{x}$ is an optimal solution and that one of the inequalities (say for $\mathbf{x}(s)$) of (9.6) is a strict inequality. Then we may replace $\mathbf{x}(s)$ with the right-hand side of this specific inequality and get a better solution. Thus, an optimal solution verifies (9.6) and from the second claim, $\boldsymbol{\pi}_{\min}$ is the unique solution of this problem. □

So, evaluation consists of solving this linear programming problem.

Complexity. By construction, the algorithm has a linear complexity with respect to the size of the formula. The complexity as a function of the size of the process depends on operators. For non-probabilistic operators, the above description should convince the reader that the complexity is again linear. For probabilistic operators, the most costly step is the resolution of a linear programming problem whose time complexity is polynomial with interior point methods [ROO 97].

9.5.4. *Overview of verification of Markov decision processes*

Markov decision processes were introduced mainly for modeling and solving optimization problems [PUT 94]. One of the pioneering works ([COU 95]) on MDP model checking refers to probabilistic satisfaction of linear temporal logic properties, and establishes exact complexity bounds for evaluation of formulae in (propositional) LTL or expressed as Büchi automata. The approach of the previous section has been extended to various branching-time logics: pCTL* [BIA 95], a variant of CTL*, and pTL and pTL* introducing observation clocks [ALF 97]. Another extension concerns logic semantics. For instance, going back to our introductory example, the modeler does not wish to take all schedulers into account. Indeed, a "real life" scheduler satisfies (even weak) fairness properties with regard to the choice of the transaction to be executed. However, the usual method which modifies the formula to take into account these hypotheses cannot be used in this context. Thus, we must introduce fair operators into the logics and design adapted algorithms [BAI 98, ALF 00]. Finally, to cope with the inaccuracy of numerical parameters of the MDP, other logics have been introduced and analyzed [ALF 04].

Temporal logic evaluation methods for MDP may also be combined with other methods to deal with more expressive models. As an example, probabilistic timed automata specify continuous-time systems where non-determinism corresponds both to the choice of the next event and its occurrence time. When these automata are non-probabilistic, the usual method builds a finite abstraction of their (usually infinite) timed transition system, which is sufficient to model check formulae of some temporal logics (such as, for instance, TCTL). This abstraction leads to several representations: region graph, zone graph, etc. In the probabilistic case, abstraction is also feasible, but the result is then an MDP which is analyzed with methods explained above [KWI 01, KWI 02b, KWI 02a]. Other more expressive models can only be analyzed with approximation techniques [KWI 00].

9.5.5. *The PRISM tool*

PRISM (Probabilistic Symbolic Model Checker) is the reference tool for model checking of probabilistic systems, developed at Birmingham University, UK, since the beginning of 2000, and available under the GPL Licence. It allows us to analyze discrete-time systems (Markov chains and Markov decision processes, PCTL logic) and continuous-time systems (Markov chains, CSL logic). PRISM usage is standard: model definition with a specific formalism, definition of the properties with probabilistic temporal logic, then computation and analysis of the results. We focus here on discrete-time system functionalities, functionalities for continuous-time systems being connected to those of the ETMCC tool presented above.

9.5.5.1. *Language of system models*

System descriptions in PRISM are based on reactive modules, close to the Alur and Henzinger [ALU 99] model used in the CTL model checking tool SMV. Modules correspond to active entities of the system and a state is defined as values of (global) variables declared and handled by modules. Synchronization is carried out by guards on values of variable. Stochastic parameters (transition probabilities) are defined in modules and the user indicates in the model if this is a DTMC or a MDP. Extending the basic model, state and transition reward functions may be defined in PRISM; it is then possible to get mean performance indices (cumulated profits or losses), in finite time horizon or in steady-state.

9.5.5.2. *Properties language*

For DTMC and MDP, PRISM allows us to define properties to be checked in the PCTL logic presented above. PRISM also computes the probability that a given PCTL formula is satisfied by all the states of a DTMC, without prefixing a probability threshold.

9.5.5.3. *Computed results*

The software analyzes the model in accordance with requested properties and returns results in the user interface, in the form of a set of graphical representations giving a syntactic view of the properties of the system. Moreover, we get mean parameters defined by reward functions of the model.

9.5.5.4. *Software architecture*

From the internal point of view, implementation is programmed with C++ for numerical code and with Java for the user interface and the overall structure of the software. An important feature of PRISM is the management of symbolic data structures based on multi-valued binary decision diagrams (MTBDD, [CLA 93]) allowing analysis of large systems (more than 10^{10} states in suitable cases). The non-probabilistic model checker systematically manages BDD structures. For numerical

solvers, PRISM uses three "engines" based on MTBDD, sparse matrix representations and a mixed model of both. Indeed, experiments and many studies carried out with the tool show that MTBDD representations tend to notably slow down numerical with respect to, now well optimized, (iterative) methods, based on sparse representations.

9.6. Bibliography

[AJM 87] AJMONE MARSAN M., CHIOLA G., "On Petri nets with deterministic and exponentially distributed firing times", ROZENBERG G., Ed., *Advances in Petri Nets 1987*, num. 266 LNCS, p. 132–145, Springer Verlag, 1987.

[AJM 95] AJMONE MARSAN M., BALBO G., CONTE G., DONATELLI S., FRANCESCHINIS G., *Modeling with Generalized Stochastic Petri Nets*, Wiley series in Parallel Computing, John Wiley & Sons, England, 1995.

[ALF 97] DE ALFARO L., "Model checking of probabilistic and non-deterministic systems", *STACS'97*, num. 1200 LNCS, Springer Verlag, p. 165–176, 1997.

[ALF 00] DE ALFARO L., "From fairness to chance", *Electronic Notes on Theoretical Computer Science*, vol. 22, 2000.

[ALF 04] DE ALFARO L., FAELLA M., HENZINGER T., MAJUMDAR R., STOELINGA M., "Model checking discounted temporal properties", *TACAS 2004: 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, num. 2988 LNCS, Springer Verlag, p. 77–92, 2004.

[ALU 99] ALUR R., HENZINGER T. A., "Reactive modules", *Formal Methods in System Design*, vol. 15, num. 1, p. 7–48, 1999.

[AND 03] ANDOVA S., HERMANNS H., KATOEN J.-P., "Discrete-time rewards model-checked", *Formal Modeling and Analysis of Timed Systems (FORMATS 2003)*, num. 2791 LNCS, Marseille, France, Springer Verlag, p. 88–103, 2003.

[AZI 96] AZIZ A., SANWAL K., V.SINGHAL, BRAYTON R., "Verifying continuous-time Markov chains", *8th Int. Conf. on Computer Aided Verification (CAV'96)*, num. 1102 LNCS, New Brunswick, NJ, USA, Springer Verlag, p. 269–276, 1996.

[AZI 00] AZIZ A., SANWAL K., V.SINGHAL, BRAYTON R., "Model checking continuous-time Markov chains", *ACM Transactions on Computational Logic*, vol. 1, num. 1, p. 162–170, 2000.

[BAI 98] BAIER C., KWIATKOWSKA M., "Model checking for a probabilistic branching-time logic with fairness", *Distributed Computing*, vol. 11(3), p. 125–155, 1998.

[BAI 03a] BAIER C., HAVERKORT B., HERMANNS H., KATOEN J.-P., "Model-checking algorithms for continuous time Markov chains", *IEEE Transactions on Software Engineering*, vol. 29, num. 7, p. 524–541, July 2003.

[BAI 03b] BAIER C., HERMANNS H., KATOEN J.-P., WOLF V., "Comparative branching-time semantics for Markov chains", *Concurrency Theory (CONCUR 2003)*, num. 2761 LNCS, Marseille, France, Springer Verlag, p. 492–507, 2003.

[BIA 95] BIANCO A., DE ALFARO L., "Model checking of probabilistic and non-deterministic systems", *FST TCS 95: Foundations of Software Technology and Theoretical Computer Science*, num. 1026 LNCS, Bangalore, India, Springer Verlag, p. 499–513, 1995.

[CAM 94] CAMPOS J., COLOM J., JUNGNITZ H., SILVA M., "Approximate throughput computation of stochastic marked graphs", *IEEE Transactions on Software Engineering*, vol. 20, num. 7, p. 526–535, July 1994.

[CHI 93a] CHIOLA G., ANGLANO C., CAMPOS J., COLOM J., SILVA M., "Operationnal analysis of timed Petri nets and application to computation of performance bounds", *Proc. of the 5th International Workshop on Petri Nets and Performance Models*, Toulouse, France, IEEE Computer Society Press, p. 128–137, October 19–22 1993.

[CHI 93b] CHIOLA G., DUTHEILLET C., FRANCESCHINIS G., HADDAD S., "Stochastic well-formed colored nets and symmetric modeling applications", *IEEE Transactions on Computers*, vol. 42, num. 11, p. 1343–1360, November 1993.

[CLA 93] CLARKE E., FUJITA M., MCGEER P., MCMILLAN K., YANG J., ZHAO X., "Multi-terminal binary decision diagrams: an efficient data structure for matrix representation", *Proc. Int. Workshop on Logics Synthesis (IWLS'93)*, p. 1–15, 1993, also available in Formal Methods in System Design, 10(2/3): 149–169, 1997.

[COU 95] COURCOUBETIS C., YANNAKAKIS M., "The complexity of probabilistic verification", *Journal of the ACM*, vol. 42(4), p. 857–907, July 1995.

[COU 03] COUVREUR J.-M., SAHEB N., SUTRE G., "An optimal automata approach to LTL model checking of probabilistic systems", in *Proc. 10th Int. Conf. Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'2003)*, num. 2850 LNAI, Almaty, Kazakhstan, Springer Verlag, p. 361–375, September 2003.

[DON 98] DONATELLI S., HADDAD S., MOREAUX P., "Structured characterization of the Markov chains of phase-type SPN", *Proc. of the 10th International Conference on Computer Performance Evaluation. Modeling Techniques and Tools (TOOLS'98)*, num. 1469 LNCS, Palma de Mallorca, Spain, Springer Verlag, p. 243–254, September 14–18 1998.

[EME 80] EMERSON E. A., CLARKE E. M., "Characterizing correctness properties of parallel programs using fixpoints", *7th International Colloquium on Automata, Languages and Programming, (ICALP)*, Noordweijkerhout, The Netherlands, p. 169–181, 1980.

[FEL 68] FELLER W., *An Introduction to Probability Theory and its Applications. Volume I*, John Wiley & Sons, 1968 (third edition).

[FEL 71] FELLER W., *An Introduction to Probability Theory and its Applications. Volume II*, John Wiley & Sons, 1971 (second edition).

[FLO 85] FLORIN G., NATKIN S., "Les réseaux de Petri stochastiques", *TSI*, vol. 4, num. 1, p. 143–160, 1985.

[GER 99] GERMAN R., TELEK M., "Formal relation of Markov renewal theory and supplementary variables in the analysis of stochastic Petri nets", BUCHHOLZ P., SILVA M., Eds., *Proc. of the 8th International Workshop on Petri Nets and Performance Models*, Zaragoza, Spain, IEEE Computer Society Press, p. 64–73, September 8–10 1999.

[GIR 03] GIRAULT C., VALK R., Eds., *Petri Nets for Systems Engineering*, Springer Verlag, 2003.

[HAD 05] HADDAD S., MOREAUX P., SERENO M., M.SILVA, "Product-form and stochastic Petri nets: a structural approach", *Performance Evaluation*, vol. 59/4, p. 313–336, 2005.

[HAV 95] HAVERKORT B., "Matrix-geometric solution of infinite stochastic Petri nets", *Proc. of the International Performance and Dependability Symposium*, IEEE Computer Society Press, p. 72–81, 1995.

[HEN 90] HENDERSON W., PEARCE C., TAYLOR P., VAN DIJK N., "Closed queueing networks with batch services", *Queueing Systems*, num. 6, p. 59–70, 1990.

[HIL 96] HILLSTON J., *A Compositional Approach to Performance Modeling*, Cambridge University Press, 1996.

[JEN 53] JENSEN A., "Markov chains as an aid in the study of Markov processes", *Skand. Aktuarietidskrift*, vol. 3, p. 87–91, 1953.

[KLE 75] KLEINROCK L., *Queueing Systems. Volume I: Theory*, Wiley-Interscience, New York, 1975.

[KLE 76] KLEINROCK L., *Queueing Systems. Volume II: Computer Applications*, Wiley-Interscience, New York, 1976.

[KWI 00] KWIATKOWSKA M., NORMAN G., SEGALA R., SPROSTON J., "Verifying quantitative properties of continuous probabilistic timed automata", *CONCUR 2000 – Concurrency Theory*, num. 1877 LNCS, Springer Verlag, p. 123–137, August 2000.

[KWI 01] KWIATKOWSKA M., NORMAN G., SPROSTON J., "Symbolic computation of maximal probabilistic reachability", *Proc. 13th International Conference on Concurrency Theory (CONCUR'01)*, num. 2154 LNCS, Springer Verlag, p. 169–183, August 2001.

[KWI 02a] KWIATKOWSKA M., NORMAN G., SEGALA R., SPROSTON J., "Automatic verification of real-time systems with discrete probability distributions", *Theoretical Computer Science*, vol. 282, p. 101–150, June 2002.

[KWI 02b] KWIATKOWSKA M., NORMAN G., SPROSTON J., "Probabilistic model checking of the IEEE 802.11 wireless local area network protocol", *Proc. PAPM/PROBMIV'02*, num. 2399 LNCS, Springer Verlag, p. 169–187, July 2002.

[LAP 95] LAPRIE J., Ed., *Guide de la sûreté de fonctionnement*, Cépaduès – Éditions, Toulouse, France, 1995.

[LIN 98] LINDEMANN C., *Performance Modeling with Deterministic and Stochastic Petri Nets*, John Wiley & Sons, New York, 1998.

[LIN 99] LINDEMANN C., REUYS A., THÜMMLER A., "DSPNexpress 2000 performance and dependability modeling environment", *Proc. of the 29th Int. Symp. on Fault Tolerant Computing*, Madison, Wisconsin, June 1999.

[MEY 80] MEYER J., "On evaluating the performability of degradable computing systems", *IEEE Transactions on Computers*, vol. 29, num. 8, p. 720–731, August 1980.

[MOL 81] MOLLOY M. K., "On the integration of delay and throughput in distributed processing models", PhD thesis, University of California, Los Angeles, CA, USA, September 1981.

[PUT 94] PUTERMAN M., *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, John Wiley & Sons Inc., New York, 1994.

[ROO 97] ROOS C., TERLAKY T., VIAL J.-P., *Theory and Algorithms for Linear Optimization. An Interior Point Approach*, Wiley-Interscience, John Wiley & Sons Ltd, West Sussex, England, 1997.

[STE 94] STEWART W. J., *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, Princeton, NJ, USA, 1994.

[TRI 82] TRIVEDI K. S., *Probability & Statistics with Reliability, Queueing, and Computer Science Applications*, Prentice Hall, Englewood Cliffs, NJ, USA, 1982.

[TRI 92] TRIVEDI K. S., MUPPALA J. K., WOOLE S. P., HAVERKORT B. R., "Composite performance and dependability analysis", *Performance Evaluation*, vol. 14, num. 3–4, p. 197–215, February 1992.

[VAR 85] VARDI M., "Automatic verification of probabilistic concurrent finite-state programs", *FOCS 1985*, p. 327–338, 1985.

[YOU 06] YOUNES H., KWIATKOWSKA M., NORMAN G., PARKER D., "Numerical vs. statistical probabilistic model checking", *Int. Journal on Software Tools for Technology Transfer (STTT)*, vol. 8, num. 3, p. 216–228, 2006.