On Probabilistic Computation Tree Logic

Frank Ciesinski, Marcus Größer {ciesinsk|groesser}@cs.uni-bonn.de

Rheinische Friedrich Wilhelms Universität zu Bonn, Institut für Informatik - Abt. I,
Römerstraße 164, 53117 Bonn

Abstract. In this survey we motivate, define and explain model checking of probabilistic deterministic and nondeterministic systems using the probabilistic computation tree logics $PCTL$ and $PCTL^*$. Juxtapositions to non-deterministic computation tree logic are made, algorithms are presented.

Keywords: $PCTL$, $PCTL^*$, Discrete Time Markov Chains, Markov Decision Processes, scheduler, fairness, Deterministic Probabilistic Systems, Nondeterministic Probabilistic Systems, quantitative model checking.

1 Introduction and Overview

Model Checking is a *fully automatic* verification method for both hard- and software systems that has seen a very pomising development for now almost 25 years. Numerous efficent methods have been proposed very successfully to attack the discouraging general complexity of Model Checking (see e.g. [12]). Various temporal logics were proposed to emphasize certain different aspects of models to be checked. Among them are logics to cover the behaviour of probabilistic systems. Taking probabilities into account in addition to nondeterministic behaviour expands the possibilities of modeling certain aspects of the system under consideration. While nondeterministic systems are considered in connection to underspecification, interleaving of several processes and interaction with the specified system from the outside, the probabilities can be exploited to model a certain probability of error or other stochastic behaviour both occuring in various real world applications, e.g. certain randomized algorithms or communication protocols over faulty media.
The development of probabilistic Model Checking can roughly be divided into two periods. With *qualitative* Model Checking, whichs development merely took place in the 1980s, it is possible to decide whether a system satisfies its specification with probability 1. Many aspects of qualitative probabilistic Model Checking were outworked, such as the termination of probabilistic programs [27], the omega-automaton-based appoach for Model Checking probabilistic (as well as non-probabilistic) systems [50, 52, 13], the tableaux based method [41] and Model Checking of probabilistic real-time systems [2]. In the 1990s the idea of qualitative Model Checking was extended to *quantitative* Model Checking where it becomes possible not only to check whether the specification is satisfied with

probability 1 but to specify arbitrary probability values $p \in [0,1]$. Model Checking algorithms were developed for the deterministic case (Discrete Time Markov Chains, $DTMCs$) [25, 3] as well as for the nondeterministic case (Markov Decision Processes, $MDPs$) [14, 15, 17, 18, 10, 8], including fairness [4].
In this survey we summarize the main results presented in the papers mentioned above and use the logic $PCTL$ (Probabilistic Computation Tree Logic), which is commonly used corresponding to Discrete Markov Chains and Markov Decision Processes as the basic temporal logic [26, 10]. It extends the temporal logic $RTCTL$ (Real Time CTL, [19]). While $RTCTL$, as an extension to the common known temporal logic CTL, additionally provides discrete time properties such as

A property holds within a certain amount of (discrete) time ,

$PCTL$ allows to state properties such as

A property holds within a certain amount of (discrete) time with a probability of at least 99%.

Thus, $PCTL$ supplies a formalism which adds probabilities in a certain way. Intuitively $RTCTL$ allows it to state "hard deadlines" while $PCTL$ is capable of stating so called "soft deadlines".
After making some basic definitions concerning probabilistic processes , the syntax and semantics of $PCTL$ and $PCTL^*$ are defined, similarities and differences between $PCTL$ and CTL (resp. $PCTL^*$ and CTL^*) are outlined. Then for both the deterministic and the nondeterministic case model checking procedures are described, fairness is taken into concern as well.
The reader is supposed to be familiar with temporal logics and basic notions of probability theory. See e.g. [22, 21, 38, 12].

2 Validation of Deterministic Probabilistic Systems

$PCTL$ - like it is presented in this chapter - was introduced by *Hansson* and *Jonsson* [25] and concentrates on probabilistic model checking in connection with probabilistic deterministic systems (PDS). In essence a PDS is a state labelled markov chain and $PCTL$ allows to check properties of the kind "starting from a state s, a certain $PCTL$ path formula holds with a probability of at least p" using a probabilistic operator. This probabilistic operator may be seen as a counterpart to the $\forall$ and $\exists$ operator in CTL (i.e. it can be applied to paths-formulae only). Similar to CTL as well, $PCTL$ can be extended by relaxing the $PCTL$-syntax to $PCTL^*$, which will be dealt with in section 2.5. We will firstly define essential structures for modeling probabilistic processes and the syntax and semantics of $PCTL$. As descriptions go along, certain similarities and differences to CTL are emphasized. Finally a model checking algorithm for $PCTL$ is outlined.

2.1 Deterministic Probabilistic Systems

We define Probabilistic Deterministic Systems (PDS) which essentially are state labelled *Finite Markov Chains* (we restrict ourselves to finite systems).

Definition 1. [Probability distribution]
Given a finite set S, a probability distribution on S is a function

$$\mu : S \to [0,1] \quad \textit{such that} \quad \sum_{s \in S} \mu(s) = 1.$$

Given a probability distribution on S, $supp(\mu)$ denotes the support, i.e. the states s of S with $\mu(s) > 0$. For $s \in S$, μ_s^1 denotes the unique distribution on S that satisfies $\mu_s^1(s) = 1$. With $Distr(S)$ we denote the set of all probability distributions on S.

Definition 2. [Finite markov chain]
A finite markov chain is a tupel

$$M = (S, T, p),$$

where

- *S is a finite set,*
- *$T \subseteq S \times S$ is a set of transitions and*
- *$p : S \times S \to [0,1]$ is a function such that for all $s \in S$*
  - *$p(s,.)$ is a probability distribution on S*
  - *$(s,t) \in T$ iff $t \in supp(p(s,.))$.*

  Throughout this article we will use the notation p_{uv} instead of $p(u,v)$.

We also refer to (S,T) as the *underlying graph* of M. Note that T is total, so (S,T) has no terminal nodes.
Normally a Markov Chain is equipped with a starting probability distribution. A markov chain with a starting distribution induces a stochastic process on the set S of its states in a natural way. The probability to start in a certain state with step 0 is determined by the starting distribution. Moreover, being in state u in the $(n-1)$th step, the probability of reaching state v with the nth step is equal to p_{uv}. The fact, that those probabilities do not depend on the previous steps (*history-independent* or *memoryless*) is called the *Markov Property.*

Definition 3. [Ergodic set]
Let $G = (V,E)$ be a directed graph. We call $C \subseteq V$ an ergodic set of G, iff C is a terminal strongly connected component of G, i.e.

- *$\forall\ (u,v) \in E\ :\ u \in C\ \Rightarrow v \in C$*
- *$\forall\ u,v \in C\ :\ \exists$ a finite path from u to v in G*

An ergodic set is therefore a strongly connected component that cannot be left once the execution sequence reached one of its states. Given a markov chain $M = (S,T,p)$, we call $C \subseteq S$ an ergodic set of M, if C is an ergodic set of the underlying graph of M.

Definition 4. [Probabilistic deterministic system (PDS)]
A probabilistic deterministic system (PDS) is a tupel

$$\mathcal{T}_{PDS} = (M, AP, L, \bar{s}),$$

where

- $M = (S, T, p)$ *is a finite markov chain,*
- AP *is a finite set of atomic propositions,*
- L *is a labelling function* $L : S \to 2^{AP}$ *that labels any state* $s \in S$ *with those atomic propositions that are supposed to hold in* s *and*
- $\bar{s} \in S$ *is the starting state.*

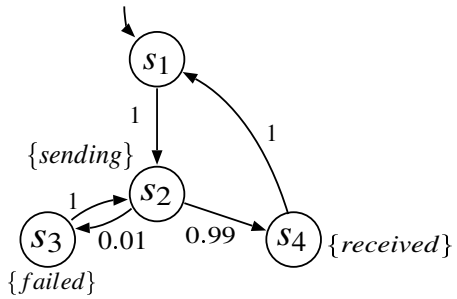


Fig. 1. PDS example: A very simple communication protocol.

In the sequel we just write $\mathcal{T}$ instead of $\mathcal{T}_{PDS}$. We now define the essential executional issues of a Probabilistic Deterministic System. Given a PDS $\mathcal{T} = (M, AP, L, \bar{s})$ with $M = (S, T, p)$ we define the following :

Definition 5. [Path]
A path of $\mathcal{T}$ *is a finite or infinite sequence* $\pi = s_0, s_1, ...$ *of states, such that* $(s_i, s_{i+1}) \in T$ *for all* i *under consideration.*
Given a finite path $\omega = s_0, s_1, \ldots, s_n$, *we denote* s_0 *by* $first(\omega)$ *and* s_n *by* $last(\omega)$. *The length* $|\omega|$ *of* ω *is equal to* n. *For an infinite path* π, *the length is equal to* ∞. *Given a path* $\pi = s_0, s_1, \ldots$ *(finite or infinite) and* $i \leq |\pi|$, *we denote the* i*-th state of* π *by* π^i *(i.e.* $\pi^i = s_i$*) and the* i*-th prefix by* $\pi{\uparrow}^i = s_0, s_1, \ldots, s_i$.
Given an infinite path $\pi = s_0, s_1, \ldots$, *we denote the suffix starting at* π^i *by* $\pi{\uparrow}_i$, *i.e.* $\pi{\uparrow}_i = s_i, s_{i+1}, \ldots$. *Furthermore we denote by* $Paths_{fin}$ *(resp.* $Paths_{inf}$*) the set of finite (resp. infinite) paths of a given* PDS *and by* $Paths_{fin}(s)$ *(resp.* $Paths_{inf}(s)$*) the set of finite (resp. infinite) paths of a given* PDS *starting at the state* s.

Definition 6. [Trace]
We define the trace of a (finite) path $\pi = s_0, s_1, \ldots$ *to be the (finite) word over*

the alphabet 2^{AP} *which we get from the following :*

$$trace(\pi) = L(\pi^0), L(\pi^1), ... = L(s_0), L(s_1), ...$$

Definition 7. [Basic cylinder]
For $\omega \in Paths_{fin}$ *the* basic cylinder *of* ω *is defined as*

$$\Delta(\omega) = \{\pi \in Paths_{inf} \quad | \quad \pi\uparrow^{|\omega|} = \omega\}.$$

Let $s \in S$. Following markov chain theory and measure theory [33] we now define a probability space on the set of paths starting in s.

Definition 8. [Probability space of a markov chain]
Given a finite markov chain $M = (S,\ T,\ p)$ *and* $s \in S$*, we define a probability space*

$$\Psi^s = (\Delta(s), \Delta^s, prob_s),$$

such that

- Δ^s *is the* σ*-algebra generated by the empty set and the basic cylinders over* S *that are contained in* $\Delta(s)$*.*
- $prob_s$ *is the uniquely induced probability measure which satisfies the following:*
  $prob_s(\Delta(s)) = 1$ *and for all basic cylinders* $\Delta(s, s_1, \ldots, s_n)$ *over* S *:*

$$prob_s(\Delta(s, s_1, \ldots, s_n)) = p_{ss_1} \cdots p_{s_{n-1}s_n}$$

We now state a fact about ergodic sets that we will need later.

Lemma 1. [Reaching an Ergodic set]
Given a markov chain $M = (S, T, p)$*, the following holds for all* $s \in S$*:*

$$prob_s(\{\pi \in Paths_{inf}(s) \mid \exists C \text{ ergodic set of } M \text{ s.t.} \forall c \in C : \pi^i = c$$

$$\text{for infinitely many } i\text{'s}\}) = 1.$$

Proof. see [33]

This means that given a markov chain $M = (S, T, p)$ and an arbitrary starting state $\bar{s}$ it holds that the stochastic process described by M and $\bar{s}$ will reach an ergodic set C of M and visit each state of C infinitely often with probability 1. This concludes the preliminary definitions needed for $PCTL$-model checking. We now define the syntax of $PCTL$.

2.2 The Syntax of *PCTL*

We define the $PCTL$ syntax along with some intuitive explanations about the corresponding semantics when meaningful. The precise semantics of $PCTL$ will be defined in a formal way in section 2.3 afterwards.

Definition 9. [*PCTL*-syntax]
The syntax of PCTL is defined by the following grammar: [1]

PCTL-state formulae

$$\Phi ::= true \;\Big|\; a \;\Big|\; \Phi_1 \wedge \Phi_2 \;\Big|\; \neg\Phi \;\Big|\; [\phi]_{\bowtie p}$$

PCTL-path formulae:

$$\phi ::= \Phi_1 \mathcal{U}^{\leq t} \Phi_2 \;\Big|\; \Phi_1 \mathcal{U} \Phi_2 \;\Big|\; \mathcal{X}\Phi$$

where $t \in \mathbb{N}$, $p \in [0,1] \subset \mathbb{R}$, and $a \in AP, \bowtie \in \{>, \geq, \leq, <\}$

The set of all *PCTL*-formulae is $form_{PCTL} = \{f : f \text{ is a } PCTL \text{ formula}\}$ where we assume a fixed set of atomic propositions. The special symbol $\bowtie$ is used for ">, $\geq$", "<" or "$\leq$" respectively.
Other Boolean operators (i.e. $\vee, \rightarrow, \leftrightarrow$) are not defined explicitely but can be derived from $\wedge$ and $\neg$ as usual. State formulae represent properties which can be any atomic proposition (i.e. $a \in AP$) or Boolean combinations of them as well as $[\ ]_{\bowtie p}$ properties that require a certain amount (w.r.t. a measure, see definition 8) of paths to be existing starting in the current state and satisfying the enclosed path formula. Path formulae involve the strong until (be it the bounded $\mathcal{U}^{\leq t}$ or the unbounded $\mathcal{U}$) or the nextstep operator $\mathcal{X}$. The nextstep can neither be derived from $\mathcal{U}$ nor from $\mathcal{U}^{\leq t}$ and neither can the bounded or unbounded until be derived from $\mathcal{X}$ [2]. Thus, we define $\mathcal{U}^{\leq t}$, $\mathcal{U}$ and $\mathcal{X}$ explicitely as three independent operators (and we will later give three different algorithms to cover these operators). Intuitively the path formula $\Phi_1 \mathcal{U}^{\leq t} \Phi_2$ states that Φ_1 holds continuously from now on until within at most t time units Φ_2 becomes true. The unbounded operator $\Phi_1 \mathcal{U} \Phi_2$ does not require a certain bound but nevertheless requires Φ_2 to become true *eventually*.
However, the weak until ($\widetilde{\mathcal{U}}$) operator can be derived by using $\mathcal{U}$ via

$$\begin{aligned}
[\Phi_1 \widetilde{\mathcal{U}}^{\leq t} \Phi_2]_{\geq p} &:= \neg[(\neg\Phi_2)\mathcal{U}^{\leq t}(\neg\Phi_1 \wedge \neg\Phi_2)]_{>(1-p)},\\
[\Phi_1 \widetilde{\mathcal{U}}^{\leq t} \Phi_2]_{>p} &:= \neg[(\neg\Phi_2)\mathcal{U}^{\leq t}(\neg\Phi_1 \wedge \neg\Phi_2)]_{\geq(1-p)},\\
[\Phi_1 \widetilde{\mathcal{U}}^{\leq t} \Phi_2]_{\leq p} &:= \neg[(\neg\Phi_2)\mathcal{U}^{\leq t}(\neg\Phi_1 \wedge \neg\Phi_2)]_{<(1-p)},\\
[\Phi_1 \widetilde{\mathcal{U}}^{\leq t} \Phi_2]_{<p} &:= \neg[(\neg\Phi_2)\mathcal{U}^{\leq t}(\neg\Phi_1 \wedge \neg\Phi_2)]_{\leq(1-p)}.
\end{aligned}$$

[1] We identify indexed terminal symbols (here: Φ_1 and Φ_2) with terminal symbols Φ in order to simplify the *PCTL*-grammar. The symbol a represents arbitrary atomic propositions. We assume $p \in [0,1]$ and $t \in \mathbb{N}$. Hence this is an abstract grammar.

[2] Informally speaking $[\Phi_1 \mathcal{U}^{\leq t} \Phi_2]_{\bowtie p, p \neq 0}$ cannot be expressed as nested $\mathcal{X}$ expressions because every $\mathcal{X}$ would have to be enclosed by another $[\ ]_{\bowtie p'}$, which cannot be done because each p, p', etc. must be constant. Furthermore it is true that $\neg\Phi \wedge [true \mathcal{U}^{\leq 1} \Phi]_{\bowtie p} \not\equiv [\mathcal{X}\Phi]_{\bowtie p}$. Nextstep states only about the future, until about the present *and* the future.

The weak until $\widetilde{\mathcal{U}}$ does not require Φ_2 to become *true*. Either Φ_1 holds for at least the next t time units or Φ_2 becomes *true* in the next t time units and Φ_1 holds continuously until this happens. The unbounded weak until operator can be defined similarly.
Similar to "always $\Box$" and "eventually $\Diamond$" operators in CTL there are pendents in $PCTL$.

$$\begin{aligned}[\Diamond^{\leq t}\Phi]_{\bowtie p} &:= [true\mathcal{U}^{\leq t}\Phi]_{\bowtie p},\\ [\Box^{\leq t}\Phi]_{\bowtie p} &:= \neg[true\mathcal{U}^{\leq t}\neg\Phi]_{\bowtie(1-p)}.\\ &= \neg[\Diamond^{\leq t}\neg\Phi]_{\bowtie(1-p)}.\end{aligned}$$

Again, the definitions are the same for the unbounded versions of these operators. The main difference between CTL and $PCTL$ lies in the extended ability to quantify over paths. While in CTL $\forall\phi$ indicates that the path formula ϕ must hold for all paths a $PCTL$ formula like $[\phi]_{>0.99}$ requires the set of paths starting in the current state satisfying ϕ to have the probability measure of more than 0.99. The formula $[\Phi_1\mathcal{U}^{\leq 10}\Phi_2]_{>0.5}$ states that the set of paths in which Φ_1 will hold continously until in at most 10 time units Φ_2 will eventually become *true* has the probability measure of more than 0.5. Some authors write statements like $\Phi_1\mathcal{U}^{\leq t}_{>p}\Phi_2$ instead of $[\Phi_1\mathcal{U}^{\leq t}\Phi_2]_{>p}$ for the sake of convenience. We omit this for the sake of a clear notation since the $[\ ]_{\bowtie p}$ quantifier can be applied to a every path formula and not only to $\mathcal{U}$ or $\widetilde{\mathcal{U}}$.

Remark 1. $PCTL$ versus $pCTL$
The syntax and semantics of $PCTL$ as defined above follow the work of *Hansson* and *Jonsson* [25]. In some papers (for instance [10]) the slightly different language $pCTL$ is used instead of $PCTL$. In $pCTL$ there is no bounded until-operator. To keep this survey consistent we present all results for the bounded until operator $\mathcal{U}^{\leq t}$ as well.

Furthergoing relations between the CTL-$\forall$ and -$\exists$ quantifier and the $PCTL$-$[\ ]_{\bowtie p}$ operator will be given after the formal definition of $PCTL$ semantics.

2.3 The Semantics of *PCTL*

While specifications written in CTL or $RTCTL$ syntax are interpreted over transitions systems (also known as Kripke structures), $PCTL$ formulae (and later $PCTL^*$ formulae) are considered in the context of probabilistic systems, be they *deterministic* (PDS) or *nondeterministic* (PNS). First of all we attend the deterministic case. For a PDS $\mathcal{T}$ we define the satisfaction relation $\models_{PDS}$, that is $\models_{PDS}\subseteq (S_{\mathcal{T}} \cup Paths_{inf}) \times form_{PCTL}$, where $S_{\mathcal{T}}$ denotes the state space of $\mathcal{T}$.

Definition 10. [$PCTL$**-semantics for** PDS**]**
Let $\models_{PDS}$ *be the smallest relation s.t. the following constraints are satisfied. We simply write* $s \models \Phi$ *instead of* $(s,\Phi) \in \models_{PDS}$ *and* $\pi \models \phi$ *instead of* $(\pi,\phi) \in \models_{PDS}$.

$$
\begin{array}{ll}
s \models true & \\
s \models a & \Leftrightarrow a \in L(s) \\
s \models \Phi_1 \wedge \Phi_2 & \Leftrightarrow s \models \Phi_1 \ and \ \ s \models \Phi_2 \\
s \models \neg\Phi & \Leftrightarrow s \not\models \Phi \\
s \models [\phi]_{\bowtie p} & \Leftrightarrow prob_s(\pi \in Paths_{inf}(s) : \pi \models \phi) \bowtie p \\
\pi \models \Phi_1 \mathcal{U}^{\leq t} \Phi_2 & \Leftrightarrow \exists i \leq t : \pi^i \models \Phi_2 \ and \ \ \forall j : 0 \leq j < i : \pi^j \models \Phi_1 \\
\pi \models \Phi_1 \mathcal{U} \Phi_2 & \Leftrightarrow \exists i \in I\!N : \pi^i \models \Phi_2 \ and \ \ \forall j : 0 \leq j < i : \pi^j \models \Phi_1 \\
\pi \models \mathcal{X}\Phi & \Leftrightarrow \pi^1 \models \Phi
\end{array}
$$

We define the equivalence relation $\equiv \subset form_{PCTL} \times form_{PCTL}$ *of two PCTL-formulae* Φ_1 *and* Φ_2 *as*

$$(\Phi_1, \Phi_2) \in \equiv \ \ \textit{iff } s \models \Phi_1 \Leftrightarrow s \models \Phi_2 \textit{ for all PDS } \mathcal{T} \textit{ and for all } s \in S_{\mathcal{T}},$$

and write $\Phi_1 \equiv \Phi_2$ *instead of* $(\Phi_1, \Phi_2) \in \equiv$.

Remark 2. In order to define the satisfaction relation $\models$ it is necessary that the set $\{\pi \in Paths_{inf}(s) : \pi \models \Phi\}$ is measurable w.r.t. $prob_s$ (see definition 8). This can be shown by structural induction [50].
We now show certain connections between $PCTL$ and CTL.

$\forall$ *and* $\exists$ *in* CTL *versus* $[\ ]_{\bowtie}$ *in* $PCTL$:
Contrary to CTL the syntactical definition of $PCTL$ does neither contain an universal quantification $\forall$ nor an existential $\exists$. Nevertheless the following equivalences for CTL formulae on the left and $PCTL$ formulae on the right do hold (where a, b are atomic propositions):

(1) $\forall(\mathcal{X}a) \equiv [\mathcal{X}a]_{\geq 1}$
(2) $\exists(\mathcal{X}a) \equiv [\mathcal{X}a]_{\geq 0}$
(3) $\exists(a\mathcal{U}b) \equiv [a\mathcal{U}b]_{>0}$, but
(4) $\forall(a\mathcal{U}b) \not\equiv [a\mathcal{U}b]_{\geq 1}$.
(5) $\forall(\Box a) \equiv [\Box a]_{\geq 1}$ or more general $\forall(a\widetilde{\mathcal{U}}b) \equiv [a\widetilde{\mathcal{U}}b]_{\geq 1}$ and
(6) $\exists(\Box a) \not\equiv [\Box a]_{>0}$ and hence $\exists(a\widetilde{\mathcal{U}}b) \not\equiv [a\widetilde{\mathcal{U}}b]_{>0}$.

Proofs for (3),(4) and (6)

(3) If at least one path π exists that fulfills the until formula, the finite prefix $\pi\uparrow^i$ of π fulfills $a\mathcal{U}b$ (for some $i \in I\!N$). It is clear that the basic cylinder of $\pi\uparrow^i$ has a measure > 0 and all paths of $\Delta(\pi\uparrow^i)$ satisfy $a\mathcal{U}b$ as well.
(4) Figure 2 shows a PDS that satisfies the $PCTL$ formula but does not satisfy the CTL formula.
(6) See also figure 2.

In $PCTL$ it would be sufficient to only include $>$ and $\geq$ in the syntax, because $<$ and $\leq$ can be derived using syntactical transformations. More precisely:

(7) $[\phi]_{>p} \equiv \neg[\phi]_{\leq p}$
(8) $[\phi]_{\geq p} \equiv \neg[\phi]_{<p}$

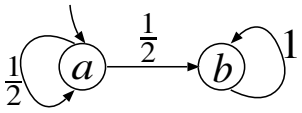


Fig. 2. PDS counterexample for (4) and (6).

The reason for including $<$ and $\leq$ in the syntax is that the transformation rules (7) and (8) become invalid for the nondeterministic case. Without too much anticipation it should be mentioned here that in order to be able to use the same $PCTL$-syntax for the deterministic case (this section) and the nondeterministic case (section 3) we include $<$ and $\leq$ but also stress the existence of (7) and (8).

2.4 Model Checking of *PCTL* against a *PDS*

The basic idea of model checking a $PCTL$ (state) formula Φ against a probabilistic deterministic system (PDS) $\mathcal{T}$ follows the idea of CTL Model Checking à la Clarke, Emerson and Sistla [11] and involves the calculation of *satisfaction sets* $Sat(\Phi)$, where $Sat(\Phi) = \{s \in S_{\mathcal{T}} : s \models \Phi\}$. In order to calculate these sets the syntax tree of Φ is constructed and the subformulas are processed in a *bottom-up* manner, more precisely the leafs are labelled with atomic propositions or *true*, while the inner nodes are labelled with $\neg\ \_$,$\_ \wedge\ \_$, $[\_\mathcal{U}^{\leq t}\_]_{\bowtie p}$,$[\_\mathcal{U}\_]_{\bowtie p}$ and $[\mathcal{X}\_]_{\bowtie p}$. Nodes labelled with $\neg\ \_$ or $[\mathcal{X}\_]_{\bowtie p}$ have exactly one son. Other inner nodes have two sons. The algorithm traverses the syntax tree in this (*postfix*) order and calculates the satisfaction sets recursively (i.e. the syntax tree is not constructed explicitely, see also figure 4).
We now present schemes for calculating solutions for the bounded and unbounded until operator followed by an algorithm that sketches a complete model checking process for $PCTL$. The calculation of a solution for the nextstep operator ($\mathcal{X}$) is presented only in the algorithm and can be understood without any further comment. See also [25].

The Bounded Until

The case $[\Phi_1\mathcal{U}^{\leq t}\Phi_2]_{\bowtie p}, p$ *arbitrary* : $P^t(s)$, as defined in the following recursion, is exactly $prob_s(\Phi_1\mathcal{U}^{\leq t}\Phi_2)$ [3] that must be calculated in order to decide whether the $PCTL$-state formula involving a certain probability is satisfied by a certain state $s \in S$.

[3] which here is used as an abbreviation for $prob_s(\{\pi \in Paths_{inf}(s) : \pi \models \Phi_1\mathcal{U}^{\leq t}\Phi_2\})$

(recursion 1)

$$P^i(s) = \begin{cases} 1, & \text{if } s \models \Phi_2 \\ 0, & \text{if } s \not\models \Phi_1 \text{ and } \ s \not\models \Phi_2, \\ 0, & \text{if } i = 0 \text{ and } \ s \not\models \Phi_2, \\ \sum_{s' \in S_{\mathcal{T}}} p_{ss'} \cdot P^{i-1}(s') & \text{otherwise.} \end{cases}$$

This recursion follows the semantical definition given in definition 10 and states a solution for the bounded until operator. Algorithm 2 performs the calculation of this recursion.

The Unbounded Until (special case)

The case $[\Phi_1 \mathcal{U} \Phi_2]_{>0}$: We will now see that also a special class of $PCTL$-formulae with *infinite* until bounds can be reduced to this scheme easily (we refer to the case $[\Phi_1 \mathcal{U} \Phi_2]_{>0}$). Later a recursion is defined whichs solutions cover arbitrary unbounded until formulae (i.e. $[\Phi_1 \mathcal{U} \Phi_2]_{\bowtie p}$). However because of equivalence (3) (see section 2.3) it is clear that also a conventional graph analysis as performed by an ordinary CTL-model checker could be used here (if Φ_1 and Φ_2 are already checked or happen to be atomic propositions). Though the use of a CTL model checker would be the preferable method in this case, we give a proof outline.

Lemma 2. $s \models [\Phi_1 \mathcal{U} \Phi_2]_{>0} \Leftrightarrow s \models [\Phi_1 \mathcal{U}^{\leq |S_{\mathcal{T}}|} \Phi_2]_{>0} \quad (\Leftrightarrow s \models \exists(\Phi_1 \mathcal{U} \Phi_2))$.

Proof. "$\Rightarrow$" If $s \models [\Phi_1 \mathcal{U} \Phi_2]_{>0}$ then $\{\pi \in paths(s) : \pi \models \Phi_1 \mathcal{U} \Phi_2\} \neq \emptyset$. It is clear that because of the finiteness of S these paths can be shortened by removing cycles such that the bounded until formula holds.
"$\Leftarrow$" obvious .

The Unbounded Until (general form)

The case $[\Phi_1 \mathcal{U} \Phi_2]_{\bowtie p}, p$ *arbitrary* : For the solution of this case the calculation schemes as presented above cannot be used. Due to the absence of a finite bound the algorithm could run infinitely and the "trick" with $\mathcal{U}^{\leq |S_{\mathcal{T}}|}$ cannot be applied since the exact calculation of the probability mass function is necessary to decide wether a certain sharp bound $p > 0$ is reached or not.
A recursion is presented [25] that is based on a partitioning of $S_{\mathcal{T}}$ into three subsets U_0, U_1 and $U_?$.

$$\begin{aligned} S_s &= Sat(\Phi_2), \\ S_f &= \{s \in S_{\mathcal{T}} : s \notin Sat(\Phi_1) \wedge s \notin Sat(\Phi_2)\}, \\ S_i &= \{s \in S_{\mathcal{T}} : s \in Sat(\Phi_1) \wedge s \notin Sat(\Phi_2)\}, \\ U_0 &= S_f \cup \{s \in S_i : \text{ there exists no path in } S_i \text{ from } s \text{ to any } s' \in S_s \ \}, \\ U_1 &= S_s \cup \{s \in S_i : \text{ the measure of reaching a state in } S_s \\ &\qquad \text{through } S_i \text{ starting in } s \text{ is } 1\} , \\ U_? &= S \backslash (U_1 \cup U_0), \end{aligned}$$

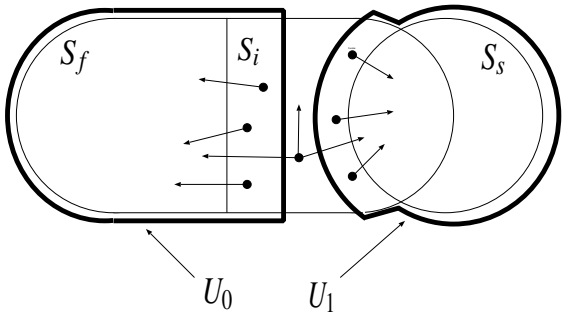


Fig. 3. The sets S_f, S_i, S_f, U_0, U_1 and $U_?$.

These sets can be calculated in the following way:

S_s, S_f, S_i**:** These sets can be calculated by investigating the labelling, respectively the satisfaction sets of the maximal proper subformulae for each state $s \in S_{\mathcal{T}}$.

Preliminary step: Remove every edge $(s, s') \in \mathcal{T}$ for each $s \in S_f \cup S_s$.

U_0**:** Perform a backward search starting in S_s. Obviously every state in $S \backslash U_0$ will be visited, which yields U_0.

U_1**:** Perform (with U_0 at hand) a backward search starting in U_0. Now every state in $S \backslash U_1$ is visited, which yields U_1.

$U_?$**:** With U_1 and U_0, calculate $U_? = S \backslash (U_1 \cup U_0)$.

With these sets the following recursion describes the measure for the unbounded until operator.

(recursion 2)

$$P^{\infty}(s) = \begin{cases} 1, & \text{if } s \in U_1 \\ 0, & \text{if } s \in U_0, \\ \sum_{s' \in S_{\mathcal{T}}} p_{ss'} \cdot P^{\infty}(s') & \text{otherwise} \end{cases}$$

This recursion is a linear equation system, which has an unique solution [25, 15].

$$x_s = \sum_{s' \in U_?} p_{ss'} \cdot x_{s'} + \sum_{s'' \in U_1} p_{ss''}, s \in U_?$$

This linear equation system can be solved in polynomial time for instance using Gaussian-elimination. However, for large probability matrices iterative methods like the Jacobi- or the Gauss-Seidel-method might be better.

Remark 3.
The until operator of CTL has a least fixpoint semantic, i.e. $Sat(\exists(\Phi_1 \mathcal{U} \Phi_2))$ and $Sat(\forall(\Phi_1 \mathcal{U} \Phi_2))$ are the least fixed points of certain monotone higher-order operators. For $PCTL$, a similar least fixed point characterization of $\mathcal{U}$ can be established (see e.g. [5]):
The probability vector $(q_s)_{s \in S_\mathcal{T}}$ where $q_s = P^\infty(s)$ is the *least solution* in $([0,1]^{|S_\mathcal{T}|})$ of the equation system

$$
\begin{aligned}
&x_s = 0 \text{ if } s \in U'_0 \\
&x_s = 1 \text{ if } s \in U'_1 \\
&x_s = \textstyle\sum_{s' \in U'_?} p_{ss'} \cdot x_{s'} + \sum_{s'' \in U'_1} p_{ss''} \text{ if } s \in U'_?
\end{aligned}
$$

where $U'_0, U'_1, U'_?$ are arbitrary sets with $U'_0 \subseteq U_0$, $U'_1 \subseteq U_1 \subseteq Sat(\Phi_2)$ and $U'_? \subseteq S_\mathcal{T} \backslash (\mathcal{U}'_0 \cup U'_1)$. In order to obtain a *unique solution* the condition $U'_0 = U_0$ is needed. The reason is that otherwise, e.g. for states $s \in S_\mathcal{T} \backslash (Sat(\Phi_1) \cup Sat(\Phi_2))$ with $p_{ss'} = 1$, we have the equation $x_s = x_s$ which, of course, has no unique solution.

The algorithm

Pseudocode for model checking $PCTL$ against a PDS is presented (algorithm 1 and algorithm 2) that uses the connections stated in the paragraphs before. In algorithm 2 we are not interested in particular if the starting state $\bar{s} \in S_\mathcal{T}$ satisfies the $PCTL$ specification. Instead the satisfaction set $Sat(\Phi)$ is calculated that contains all states $s \in S_\mathcal{T}$ *s.t.* $s \models \Phi$. In addition to that we need a stub algorithm (algorithm 1) which *is* called with the starting state and whichs only task is to check whether it is true or not that the starting state is contained in $Sat(\Phi)$ (which was calculated by algorithm 2).
In algorithm 2 it is stated (implicitely due to the recursive execution) that a syntax tree of Φ is built and processed in bottom up manner. This assures an efficient complexity for the processing of the subformulae and is illustrated in figure 4. The syntax tree is built according to the $PCTL$-grammar. Atomic propositions will always be leafs of this tree, the Sat sets are calculated for these leafs. The next inner nodes to be processed will either be Boolean combinations (i.e. $\wedge, \vee$ or $\neg$) or nextstep- or until-expressions. Sat sets for these are calculated according to the cases in the algorithm.

The Complexity of *PCTL* model checking against a *PDS*

We now outline the complexity of the model checking algorithm for checking a $PCTL$ formula Φ against a PDS. Given a PDS $\mathcal{T} = (M, AP, L, \overline{s}), M = (S, T, p)$, we give asymptotic complexity results. Obviously, the number of subformulae to be checked is $\leq length(\Phi)$.

$$\Phi = sending \rightarrow [\Diamond^{\leq 5} sent]_{>0.99}$$
$$\equiv \neg sending \vee [\Diamond^{\leq 5} sent]_{>0.99}$$

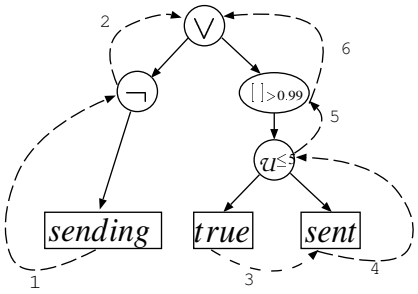


Fig. 4. Syntaxtree example.

Algorithm 1 PCTL_MODELCHECK(Φ, $\mathcal{T}$)

Input: $PDS\ \mathcal{T}, PCTL$ formula $\Phi, \bar{s} \in S_{\mathcal{T}}$
Output: truth value of $\bar{s} \models_{PDS} \Phi$

calculate $Sat(\Phi)$ with algorithm 2.
IF $s \in Sat(\Phi)$ **THEN**
    return $true$
ELSE
    return $false$

Algorithm 2 $Sat(\Phi)$

Input: $PCTL$ state formula,
Output: Set of all $s \in S$ that satisfy Φ

CASE Φ is

$true$: return $S_{\mathcal{T}}$;

a : return $\{s \in S_{\mathcal{T}} : a \in L(s)\}$;

$\Phi_1 \wedge \Phi_2$: return $Sat(\Phi_1) \cap Sat(\Phi_2)$;

$\neg\Phi'$: return $S_{\mathcal{T}} \setminus Sat(\Phi')$;

$[\mathcal{X}\Phi']_{\bowtie p}$: calculate $Sat(\Phi')$ and return $\{s \in S_{\mathcal{T}} \mid \Sigma_{s' \in Sat(\Phi')} p_{ss'} \bowtie p\}$;

$[\Phi_1 \mathcal{U}^{\leq t} \Phi_2]_{\bowtie p}$: calculate $Sat(\Phi_1)$ and $Sat(\Phi_2)$;

$$\forall s \in S_{\mathcal{T}} : P^0(s) = \begin{cases} 1 & \text{if } s \in Sat(\Phi_2), \\ 0 & \text{otherwise} \end{cases}$$

FOR $i = 1$ to t **DO**

$$\forall s \in S_{\mathcal{T}} : P^i(s) = \begin{cases} 0, \text{ if } & s \notin Sat(\Phi_1) \text{ and } s \notin Sat(\Phi_2), \\ 1, \text{ if } & s \in Sat(\Phi_2), \\ \sum_{s' \in S_{\mathcal{T}}} p_{ss'} \cdot P^{i-1}(s') & \text{otherwise .} \end{cases}$$

OD;
return $\{s \in S_{\mathcal{T}} : P^t(s) \bowtie p\}$;

$[\Phi_1 \mathcal{U} \Phi_2]_{>0}$: calculate $Sat(\Phi_1)$ and $Sat(\Phi_2)$ and do conventional graph analysis;
return $\{s \in S_{\mathcal{T}} : s \models \exists(\Phi_1 \mathcal{U} \Phi_2)\}$;

$[\Phi_1 \mathcal{U} \Phi_2]_{\bowtie p}$: calculate $Sat(\Phi_1)$ and $Sat(\Phi_2)$ and solve linear equation system;
return $\{s \in S_{\mathcal{T}} : P^{\infty}(s) \bowtie p\}$;

END CASE

- For the case that Φ' is an atomic proposition, its negation or a boolean combination of them, the satisfaction set $Sat(\Phi')$ can be computed in time $O(|S|)$.
- If Φ' is a nextstep expression $[\mathcal{X}\Phi'']_{\bowtie p}$ it can be checked in time $O(|S|+|T|)$ (where we assume a sparse matrix representation).
- The case $\Phi' = [\Phi_1 \mathcal{U}^{\leq t} \Phi_2]_{\bowtie p}$ can be checked in time $O(|V| \cdot |T| \cdot t)$.
- The case $\Phi' = [\Phi_1 \mathcal{U} \Phi_2]_{\bowtie p}$ can be checked by computing the sets U_0, U_1 and $\mathcal{U}_?$ in time $O(|S| + |T|)$ each and then solving a linear equation system of size $(|S| \times |S|)$ which is possible to do in time $O(poly(|S|))$.
- The recursion tree of algorithm 2 (wich can be seen as building and processing the systaxtree of Φ) requires $O(length(\Phi))$ steps.

Theorem 1. *Let $\mathcal{T}$ be a PNS and Φ a PCTL state formula over the set of atomic propositions of $\mathcal{T}$. Then $Sat(\Phi)$ can be computed in time $O(poly(|S|) \cdot length(\Phi))$.*

Proof. The correctness of theorem 1 follows from the statements about the complexities made before.

2.5 The Syntax and semantics of *PCTL**

We now introduce the logic $PCTL^*$ and explain how to model check a PDS against a $PCTL^*$-formula. Throughout this section let $\mathcal{T} = (M, AP, L, \bar{s})$ be a PDS with $M = (S, T, p)$.
Like standard CTL we can extend $PCTL$ to a richer logic $PCTL^*$ which allows the negation and conjunction of path formulae and also the combination of temporal modalities. In addition, every state formula is as well a path formula.

Definition 11. [PCTL*-Syntax]
Given the set of atomic propositions AP, the syntax of $PCTL^*$-formulae over AP is defined by the following grammar:

$PCTL^*$-state formulae

$$\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid [\phi]_{\bowtie p}$$

$PCTL^*$-path formulae:

$$\phi ::= \neg\phi \mid \phi_1 \wedge \phi_2 \mid \Phi \mid \phi_1 \mathcal{U}^{\leq t} \phi_2 \mid \phi_1 \mathcal{U} \phi_2 \mid \mathcal{X}\phi$$

where $a \in AP$, $t \in \mathbb{N}, p \in [0,1] \subset \mathbb{R}$ and $\bowtie \in \{\geq, >, \leq, <\}$.

With $State_{PCTL^*} = \{\Phi : \Phi \text{ is a } PCTL^* \text{ state formula}\}$ we denote the set of all $PCTL^*$ state formulae and with $Path_{PCTL^*} = \{\phi : \phi \text{ is a } PCTL^* \text{ path formula}\}$ we denote the set of all $PCTL^*$ path formulae for a fixed set AP of atomic propsitions. Observe that $State_{PCTL^*} \subset Path_{PCTL^*}$ holds.
For the PDS $\mathcal{T}$ we define the satisfaction relation $\models^*_{PDS}$, which is a relation on $(S \cup Paths_{inf}) \times Path_{PCTL^*}$.

Definition 12. [PCTL*-Semantics for Probabilistic Deterministic Systems]
Let $\models^*_{PDS}$ be the smallest relation on $(S \cup Paths_{inf}) \times Path_{PCTL^*}$ s.t. for $a \in AP$, $\Phi, \Phi_1, \Phi_2 \in State_{PCTL^*}$, $\phi, \phi_1, \phi_2 \in Path_{PCTL^*}$ and $s \in S$: [4]

$$
\begin{array}{ll}
s \models true & \\
s \models a & \Leftrightarrow a \in L(s) \\
s \models \Phi_1 \wedge \Phi_2 & \Leftrightarrow s \models \Phi_1 \wedge s \models \Phi_2 \\
s \models \neg\Phi & \Leftrightarrow s \not\models \Phi \\
s \models [\phi]_{\bowtie p} & \Leftrightarrow prob_s(\{\pi \in Paths_{inf}(s) \ : \ \pi \models \phi\}) \bowtie p \\
\pi \models \phi_1 \wedge \phi_2 & \Leftrightarrow \pi \models \phi_1 \wedge \pi \models \phi_2 \\
\pi \models \neg\phi & \Leftrightarrow \pi \not\models \phi \\
\pi \models \Phi & \Leftrightarrow \pi^0 \models \Phi \\
\pi \models \phi_1 \mathcal{U}^{\leq t} \phi_2 & \Leftrightarrow \exists \ \ i \leq t \text{ such that } \pi\uparrow_i \models \phi_2 \text{ and } \forall j : 0 \leq j < i : \pi\uparrow_j \models \phi_1 \\
\pi \models \phi_1 \mathcal{U} \phi_2 & \Leftrightarrow \exists \ \ i \in I\!N \text{ such that } \pi\uparrow_i \models \phi_2 \text{ and } \forall j : 0 \leq j < i : \pi\uparrow_j \models \phi_1 \\
\pi \models \mathcal{X}\phi & \Leftrightarrow \pi\uparrow_1 \models \phi
\end{array}
$$

Note, that the definition of the semantics requires that the set $\{\pi \in Paths_{inf}(s) : \pi \models \phi\}$ is measurable for a $PCTL^*$ formula ϕ. Like for $PCTL$ formulae, this can be shown by structural induction (see remark 2).

Remark 4.
Note, that for a $PCTL^*$-path formula the following equivalences hold, since $Paths_{inf}$ is the disjoint union of $\{\pi \in Paths_{inf} : \pi \models \phi\}$ and $\{\pi \in Paths_{inf} : \pi \models \neg\phi\}$.

- $s \models [\phi]_{<p} \quad \Leftrightarrow \quad s \models [\neg\phi]_{>(1-p)}$
- $s \models [\phi]_{\leq p} \quad \Leftrightarrow \quad s \models [\neg\phi]_{\geq(1-p)}$

Due to this fact, it is sufficient to consider formulae which do not make use of $[.]_{<p}$ and $[.]_{\leq p}$. Note however, that the above equivalences can not be applied for $PCTL$ formulae, since it is not allowed to negate $PCTL$-path formulae.

As in the nonprobabilistic case, one could also define the logic $PCTL^+$, which extends $PCTL$ by negation and conjunction of path formulae, but does not allow the combination of temporal modalities (e.g. $\phi_1 \mathcal{U} \phi_2$, where ϕ_1 and ϕ_2 are path formulae). In contrast to the nonprobabilistic case, where CTL^+ is not more expressive than CTL, we believe that $PCTL^+$ is strictly more expressive than $PCTL$. We only give an informal argument and show where the transformation method from CTL^+ to CTL fails in the probabilistic case. Consider the CTL^+ formula $\Phi = \exists(\Diamond a \wedge \Diamond b)$. Then the following equivalence holds :

$$\exists(\Diamond a \wedge \Diamond b) \quad \equiv \quad \exists\Diamond(a \wedge \exists\Diamond b) \ \vee \ \exists\Diamond(b \wedge \exists\Diamond a),$$

where the formula on the right hand side is a CTL formula. Considering only one path (due to the existential quantification), the above equivalence is easy to

[4] Observe that we use the notation $s \models \Phi$ (resp. $\pi \models \phi$) instead of $(s, \Phi) \in \models^*_{PDS}$ (resp. $(\pi, \phi) \in \models^*_{PDS}$).

accomplish. But in the probabilistic setting, we have the following problem :

$$[\Diamond a \wedge \Diamond b]_{\trianglerighteq p} \quad \not\equiv \quad [\Diamond(a \wedge [\Diamond b]_{?})]_{?} \ \vee \ [\Diamond(b \wedge [\Diamond a]_{?})]_{?},$$

whatever might be in place for the question marks. Even the following idea with an infinite disjunction of $PCTL$ formulae does not work:

$$[\Diamond a \wedge \Diamond b]_{\trianglerighteq p} \quad \not\equiv \bigvee_{p_1 \cdot p_2 + p_3 \cdot p_4 \trianglerighteq p} [\neg b \, \mathcal{U}(a \wedge [\Diamond b]_{\trianglerighteq p_2})]_{\trianglerighteq p_1} \quad \wedge \quad [\neg a \, \mathcal{U}(b \wedge [\Diamond a]_{\trianglerighteq p_3})]_{\trianglerighteq p_4}.$$

Note that the infinite disjunction is not a $PCTL$ formula, since $PCTL$ formulae are always finite (so the $\not\equiv$ is not really appropriate in this case). Moreover do we believe that there is no alternative way to express $[\Diamond a \wedge \Diamond b]_{\trianglerighteq p}$ in an equivalent $PCTL$ formula.

For the sake of completeness we shortly describe the logic LTL since we will need it later.

Definition 13. [LTL-Syntax]
Given the set of atomic propositions AP, LTL-formulae over AP are $PCTL^*$-path formulae given by the following grammar:

LTL formulae:

$$\phi ::= true \ \Big| \ a \ \Big| \ \neg\phi \ \Big| \ \phi_1 \wedge \phi_2 \ \Big| \ \phi_1 \mathcal{U}^{\leq t} \phi_2 \ \Big| \ \phi_1 \mathcal{U} \phi_2 \ \Big| \ \mathcal{X}\phi$$

where $a \in AP$ and $t \in \mathbb{N}$.

We define the language $\mathcal{L}_\omega(\phi)$ of a LTL formula ϕ over AP to be the set of infinite words w over 2^{AP} such that $w \models \phi$, assuming $L(x) = x \ \ \forall x \in 2^{AP}$. Note that for a path π the following holds :

$$\pi \models \phi \quad \text{iff} \quad trace(\pi) \in \mathcal{L}_\omega(\phi).$$

2.6 Model Checking of $PCTL^*$ against a PDS

There are different approaches to $PCTL^*$ model checking against deterministic system (see e.g. [3, 13, 51]). We follow the automata theoretic approach since we will do the same in section 3.3 where we discuss nondeterministic systems. Given a state formula $\Phi \in State_{PCTL^*}$ we want to compute the set $Sat(\Phi)$. To do so, we proceed in the same way as with CTL^* formulae (see [20]).
As with $PCTL$ formulae, we first create the parse tree of our given formula and then work from the bottom to the top using satisfaction sets for all subformulae. Having a look at the above defined semantics tells us that conjunction (resp. negation) are just the intersection (resp. the complement) of satisfaction sets of subformulae[5]. The terminal cases are also easy to handle. That leaves us with the

[5] $Sat(\Phi_1 \wedge \Phi_2) = Sat(\Phi_1) \cap Sat(\Phi_2)$ and $Sat(\neg\Phi) = S \setminus Sat(\Phi)$

case $\Phi = [\phi]_{\bowtie p}$. We assume that we have already computed the satisfaction sets for all maximal state subformulae $\gamma_1, \ldots, \gamma_n$ of ϕ and have labelled the states of S appropriately with new atomic propositions $r_1, \ldots, r_n$[6]. Let

$$\phi' = \phi \{^{\gamma_1}/_{r_1}, \ldots, ^{\gamma_n}/_{r_n}\}$$

be the formula we get from ϕ by replacing each occurence of γ_i by r_i, $1 \leq$ i $\leq$ n. Then ϕ' is a LTL formula over the atomic propositions $r_1, \ldots, r_n$. In order to decide whether $s \models^*_{PDS} [\phi']_{\bowtie p}$, we have to compute

$$prob_s(\{\pi \in Paths_{inf}(s) : \pi \models \phi'\}).$$

This can be done by using the standard ω-automaton approach (see e.g. [48, 52]). A ω-automaton accepts infinite words over a given alphabet. In the probabilistic setting a kind of determinism is needed. Here we use deterministic Rabin automata.

Definition 14. [Deterministic Rabin Automaton]
A deterministic Rabin automaton is a tupel

$$\mathcal{A} = (\Sigma, Q, \bar{q}, \rho, \alpha),$$

where

- Σ *is a finite alphabet,*
- Q *is a finite set of states,*
- $\bar{q} \in Q$ *is an initial state,*
- $\rho : Q \times \Sigma \longrightarrow Q$ *is a transition function and*
- $\alpha \subseteq 2^Q \times 2^Q$ *is an acceptance condition.*

Definition 15. [Run, Limit and α-satisfaction]
Given a deterministic Rabin automaton $\mathcal{A} = (\Sigma, Q, \bar{q}, \rho, \alpha)$ and an infinite word $w = a_1, a_2, \ldots \in \Sigma^\omega$ over Σ we call the unique sequence $r = \bar{q}, q_1, q_2, \ldots$ with

$$q_i = \rho(q_{i-1}, a_i) \qquad (q_0 = \bar{q})$$

the run of $\mathcal{A}$ over w.
We define the limit of a run $r = \bar{q}, q_1, q_2, \ldots$ as

$$lim(r) = \{q \ : \ q = q_i \ \text{ for infinitely many } i's\}.$$

We say a subset $X \subseteq Q$ satisfies α if and only if

$$\exists\ (L,\ U) \in \alpha \ : L \cap X \neq \emptyset \ \wedge \ U \cap X = \emptyset.$$

We say a run r satisfies α if and only if $lim(r)$ satisfies α.
Observe that Rabin acceptance can be expressed in a LTL formula, i.e. r satisfies α iff r satisfies the LTL formula

$$\vee_{(L,U)\in\alpha} (\Box\Diamond L \wedge \Diamond\Box\neg U).$$

[6] $r_i \in L(s)$ iff $s \in Sat(\gamma_i)$, where AP is extended by $\{r_1, \ldots, r_n\}$

We now define the language of a deterministic Rabin automaton.

Definition 16. [$\mathcal{L}_\omega(\mathcal{A})$]
Let $\mathcal{A} = (\Sigma, Q, \bar{q}, \rho, \alpha)$ be a deterministic Rabin automaton and $w \in \Sigma^\omega$. Let r be the run of $\mathcal{A}$ over w. We say $\mathcal{A}$ accepts w if and only if r satisfies α. We call the set

$$\mathcal{L}_\omega(\mathcal{A}) \ = \ \{w \in \Sigma^\omega : \mathcal{A} \text{ accepts } w\}$$

the language of $\mathcal{A}$.

We now state a useful lemma (see [48, 52, 51, 44]).

Lemma 3. [Rabin automaton for a LTL formula]
Given a LTL formula ϕ over AP, a Deterministic Rabin automaton $\mathcal{A}$ with the alphabet 2^{AP} can be constructed such that

$$\mathcal{L}_\omega(\phi) = \mathcal{L}_\omega(\mathcal{A}).$$

Remember that we want to compute the value

$$prob_s(\{\pi \in Paths_{inf}(s) \ : \ \pi \models \phi'\}),$$

where ϕ' is a LTL formula over the extended set AP of atomic propositions. We first construct a deterministic Rabin automaton $\mathcal{A} = (\Sigma, Q, \bar{q}, \rho, \alpha)$ with $\mathcal{L}_\omega(\phi') = \mathcal{L}_\omega(\mathcal{A})$.
We then build a product markov chain $\mathcal{T} \times \mathcal{A}$ for $\mathcal{T}$ and $\mathcal{A}$ by the following :
$\mathcal{T} \times \mathcal{A} = (S \times Q, \ T', \ p')$ with

- $p'((s,q),(s',q')) \ = \ p_{ss'}$ if $q' = \rho(q, L(s))$ and 0 otherwise
- $T' = \{((s,q),(s',q')) \mid p'((s,q),(s',q')) \neq 0\}$.

It is easy to see that there is a one-to-one correspondence between the paths of $\mathcal{T}$ and those of $\mathcal{T} \times \mathcal{A}$ by path-lifting. In addition, taking a path π of $\mathcal{T}$, lifting it to a path π' of $\mathcal{T} \times \mathcal{A}$ and then reducing it to its second component, gives the run of $\mathcal{A}$ over $trace(\pi)$. So $\mathcal{T} \times \mathcal{A}$ also keeps track whether $trace(\pi)$ will be accepted by $\mathcal{A}$ or not, i.e. whether π satisfies ϕ' or not. Taking this into account it is easy to show that

$$prob_s^{\mathcal{T}}(\{\pi \in Paths_{inf}(s) \mid \pi \models \phi'\}) =$$
$$prob_{(s,\bar{q})}^{\mathcal{T}\times\mathcal{A}}(\{\pi' \text{ path in } \mathcal{T} \times \mathcal{A} \mid \pi' \text{ satisfies } \alpha'\}),$$

where α' is the lifted accepted condition from $\mathcal{A}$, i.e.

$$\alpha' = \{(S \times L, \ S \times U) \mid (L, \ U) \in \alpha\}.$$

Let Acc be the union of the ergodic sets of $\mathcal{T} \times \mathcal{A}$ that satisfy α'. Using lemma 1 about ergodic sets, it follows that

$$prob_{(s,\bar{q})}^{\mathcal{T}\times\mathcal{A}}(\{\pi' \text{ path in } \mathcal{T} \times \mathcal{A} \mid \pi' \text{ satisfies } \alpha'\}) \ =$$
$$prob_{(s,\bar{q})}^{\mathcal{T}\times\mathcal{A}}(\{\pi' \text{ path in } \mathcal{T} \times \mathcal{A} \mid \pi' \text{ reaches } Acc\}).$$

The right side of this equation equals

$$prob^{\mathcal{T}\times\mathcal{A}}_{(s,\bar{q})}(\{\pi' \text{ path in } \mathcal{T}\times\mathcal{A} \mid \pi' \models \Diamond acc\}),$$

when exactly the states $\in Acc$ are labelled with acc and can therefore be computed by the algorithms introduced in the $PCTL$ section 2.4. The following flow chart (figure 5) sketches the main ideas:

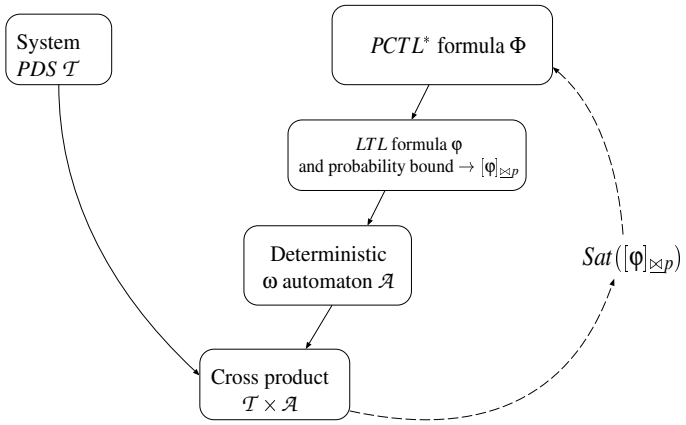


Fig. 5. Model Checking Scheme for $PCTL^*$

As a closing remark we want to mention that the size of the constructed automaton $\mathcal{A}$ is doubly exponential in the size of the formula ϕ'. And the time needed to compute $prob_s(\{\pi \in Paths_{inf}(s) \ : \ \pi \models \phi\})$ is doubly exponential in $|\phi|$. However, in [15] a different approach is discussed, which is only single exponential in $|\phi|$. In addition, both methods are polynomial in the size of the system. (See the article by B. Bollig and M. Leuker in this volume.)

3 Validation of Nondeterministic Probabilistic Systems

In section 2 methods were established to show how to model check $PCTL$ and $PCTL^*$ against a PDS. We now move on to probabilistic *nondeterministic* systems that involve a set of probability distributions per transition rather than only one as in the deterministic case. In non-probabilistic systems nondeterminism can be used to describe input behaviour from outside the system (e.g. user input) or certain fault possibilities. Also the interleaving behaviour of asynchronous parallel processes can be expressed using nondeterminism. This carries over to the

probabilistic setting, in which also underspecification can be expressed with nondeterminism. In the nondeterministic case *schedulers* are introduced that pick the distributions as the execution of a nondeterministic system goes along. These probabilistic nondeterministic systems essentially are state-labelled *Markov Decision Processes* ($MDPs$).

3.1 Nondeterministic Probabilistic Systems

We now define nondeterminism in the probabilistic context. Several types of nondeterministic systems were introduced in [50, 24, 26, 46] and [10]. We follow here the approach of [10].

Definition 17. [Probabilistic Nondeterministic System (PNS)]
A probabilistic nondeterministic system is a tupel

$$\mathcal{T}_{PNS} = (S, Steps, AP, L, \bar{s}),$$

where

- S *is a finite set of states,*
- $Steps$ *is a function that assigns to each state* $s \in S$ *a finite, non-empty set of probability distributions on* S*, i.e.*

$$Steps \ : \ S \ \rightarrow 2^{Distr(S)},$$

  such that $1 \leq |Steps(s)| < \infty \quad \forall s \in S$.
- AP *is a finite set of atomic propositions,*
- L *is a labelling function* $L : S \rightarrow 2^{AP}$ *that labels any state* $s \in S$ *with those atomic propositions that are supposed to hold in* s *and*
- $\bar{s} \in S$ *is the starting state.*

Similar to section 2.1 for the rest of this chapter we simply write $\mathcal{T}$ instead of $\mathcal{T}_{PNS}$.
Thus the main difference between definition 4 of a PDS and the definition of a PNS is, that there is not only one transition distribution per state but a set of distributions and a function $Steps$ that yields them for each state. Intuitively, the set $Steps(s)$ represents the non-deterministic alternatives in state s. Having resolved the nondeterminism in a state s, i.e. having chosen a $\mu \in Steps(s)$, the process itself resolves the probabilistic choice by selecting some state s' with the probability $\mu(s')$. We refer to the elements of $Steps(s)$ as the transitions of s.
We now explain how a path is defined in a PNS.

Definition 18. [Path in a PNS]
Let $\mathcal{T} = (S, Steps, AP, L, \bar{s})$ *be a probabilistic nondeterministic system. We define a path of* $\mathcal{T}$ *to be a (finite or infinite) sequence*

$$\pi = s_0 \ \xrightarrow{\mu_1} s_1 \ \xrightarrow{\mu_2} s_2 \ \ldots$$

s.t. for all i *under consideration* : $\ s_i \in S, \ \mu_{i+1} \in Steps(s_i)$ *and* $\mu_{i+1}(s_{i+1}) > 0$.

The definitions of the length of a path, of $first(\pi), last(\pi), \pi^k, \pi\uparrow^k$ *and* $\pi\uparrow_k$ *and of* $Paths_{fin}, Paths_{inf}$ *(resp.* $Paths_{fin}(s), Paths_{inf}(s)$*) carry over from the deterministic case (see definition 5). If* $k < |\pi|$*, we put* $step(\pi, k) = \mu_{k+1}$*. As for deterministic Rabin automata, given an infinite path* π*, we denote by* $lim(\pi)$ *the set of states that occur infinitely often in* π*.*

But how is the nondeterminism resolved? We give the definition of a scheduler (sometimes called strategy, adversary or policy) that decides, based on the past history of the system, which possible step to perform next.

Definition 19. [Scheduler]
Let $\mathcal{T} = (S, Steps, AP, L, \bar{s})$ *be a probabilistic nondeterministic system. A scheduler of* $\mathcal{T}$ *is a function* B *mapping every finite path* ω *of* $\mathcal{T}$ *to a distribution* $B(\omega) \in Steps(last(\omega))$*.*

That means, after the process described by the PNS performed the first $|\omega|$ steps, the scheduler B chooses the next transition to be $\mu = B(\omega)$, thus resolving the nondeterminism. So if a system behaves according to a scheduler B and has followed the sequence $\omega = s_0, s_1, \ldots, s_n$ so far, then it will be in state s in the next step with probability

$$B(\omega)(s).$$

We denote the set of infinite paths that comply with a given scheduler B with $Paths^B_{inf}$, i.e.

$$Paths^B_{inf} \;=\; \{\pi \in Paths_{inf} \;:\; B(\pi\uparrow^i) = step(\pi, i) \quad i = 0, 1, 2, \ldots\}.$$

We call a scheduler B

$$simple, \quad \text{iff} \quad B(\omega) = B(last(\omega)) \;\forall \text{ finite path } \omega,$$

i.e. B is history independent.
For a state s to satisfy a $PCTL^*$ state formula of the type $[\phi]_{\bowtie p}$[7], we need to ensure that starting in s, the given system produces a path satisfying ϕ with a probability $\bowtie p$ for all schedulers under consideration.
We therefore define for each $s \in S$ and for each scheduler B a probability space on the set of infinite paths starting in s. The basic cylinders of a PNS are defined exactly as the basic cylinders of a PDS, see definition 7. So we can give the definition of the needed probability spaces.

Definition 20. [Probability space of a PNS]
Given a PNS $\mathcal{T} = (S, Steps, AP, L, \bar{s})$*, a scheduler* B *and* $s \in S$*, we define a probability space*

$$\Psi^B_s = (\Delta(s), \Delta^s, prob^B_s),$$

such that

- Δ^s *is the* σ*-algebra generated by the empty set and the basic cylinders that are contained in* $\Delta(s)$*.*

[7] Remember, that $\bowtie \in \{\geq, >, \leq, <\}$.

– *$prob_s^B$ is the uniquely induced probability measure which satisfies the following : $prob_s^B(\Delta(s)) = 1$ and for all basic cylinders $\Delta(\omega) \subseteq \Delta(s)$ with $\omega = s_0, s_1, \ldots, s_n$*[8] *:*

$$prob_s^B(\Delta(\omega)) \; = prob_s^B(\Delta(s_0, s_1, \ldots, s_n)) \; = \; \prod_{i=0}^{|\omega|-1} B(s_0, \ldots, s_i)(s_{i+1}).$$

Note that the underlying sigma algebra is the same for all schedulers.

Remark 5.
To get the above probability space Ψ_s^B for some $s \in S$ and some scheduler B one can also consider the markov chain

$$M_s^B = (S_s^B, T_s^B, p_s^B),$$[9]

where

– $S_s^B = Paths_{fin} \cap \Delta(s)$ and

for all $\omega, \omega' \in S_s^B \times S_s^B$

– $p_s^B(\omega, \omega') = B(\omega)(last(\omega')), \quad$ if $\omega = \omega' \uparrow^{|\omega'|-1}$ and 0 otherwise.
– $(\omega, \omega') \in T_s^B \quad \Leftrightarrow \quad p_s^B(\omega, \omega') > 0.$

If we now identify $\omega \in Paths_{fin}$ with ω_s^B, where $(\omega_s^B)^i = \omega \uparrow^i, \; i = 0, 1, \ldots, |\omega|$ and $|\omega_s^B| = |\omega|$, we get that $prob_s^B(\Delta(\omega))$ equals the measure of $\Delta(\omega_s^B)$ in the markov chain M_s^B.

Now we have everything on hand to define the semantics of $PCTL^*$ formulae over a PNS with respect to a given set of schedulers. We follow the work of [10]. For the rest of this section let $\mathcal{T} = (S, Steps, AP, L, \bar{s})$ be a PNS and let $Sched$ be a nonempty set of schedulers of $\mathcal{T}$.

Definition 21. [PCTL*-Semantics for Probabilistic Nondeterministic Systems]
Let $\models_{Sched}$ be the smallest relation on $(S \cup Paths_{inf}) \times Path_{PCTL^*}$ s.t. for $a \in AP, \quad \Phi, \Phi_1, \Phi_2 \in State_{PCTL^*}, \quad \phi, \phi_1, \phi_2 \in Path_{PCTL^*}$ and $s \in S$:

[8] Note that $s_0 = s$.
[9] The state set of M_s^B is countable. Nevertheless do the definitions given in section 2.1 for finite markov chains also work here.

$$
\begin{array}{ll}
s \models_{Sched} true & \\
s \models_{Sched} a & \Leftrightarrow a \in L(s) \\
s \models_{Sched} \Phi_1 \wedge \Phi_2 & \Leftrightarrow s \models_{Sched} \Phi_1 \wedge s \models_{Sched} \Phi_2 \\
s \models_{Sched} \neg\Phi & \Leftrightarrow s \not\models_{Sched} \Phi \\
s \models_{Sched} [\phi]_{\bowtie p} & \Leftrightarrow prob_s^B(\{\pi \in Paths_{inf}^B(s) : \pi \models_{Sched} \phi\}) \underline{\bowtie} p \ \ \forall B \in Sched \\
\pi \models_{Sched} \phi_1 \wedge \phi_2 & \Leftrightarrow \pi \models_{Sched} \phi_1 \wedge \pi \models_{Sched} \phi_2 \\
\pi \models_{Sched} \neg\phi & \Leftrightarrow \pi \not\models_{Sched} \phi \\
\pi \models_{Sched} \Phi & \Leftrightarrow \pi^0 \models_{Sched} \Phi \\
\pi \models_{Sched} \phi_1 \mathcal{U}^{\leq t} \phi_2 & \Leftrightarrow \exists \ \ i \leq t \text{ such that } \pi\uparrow_i \models_{Sched} \phi_2 \text{ and} \\
 & \quad \forall j : 0 \leq j < i : \pi\uparrow_j \models_{Sched} \phi_1 \\
\pi \models_{Sched} \phi_1 \mathcal{U} \phi_2 & \Leftrightarrow \exists \ \ i \in I\!N \text{ such that } \pi\uparrow_i \models_{Sched} \phi_2 \text{ and} \\
 & \quad \forall j : 0 \leq j < i : \pi\uparrow_j \models_{Sched} \phi_1 \\
\pi \models_{Sched} \mathcal{X}\phi & \Leftrightarrow \pi\uparrow_1 \models_{Sched} \phi
\end{array}
$$

Again it is required, that the set $\{\pi \in Paths_{inf}^B \ : \ \pi \models_{Sched} \phi\}$ is measurable for a $PCTL^*$ formula ϕ and a scheduler B, which follows from the analogical fact in the deterministic setting and remark 5 (or can be shown by structural induction).
So in order to model check a formula of the kind $[\phi]_{\underline{\bowtie} p}$ for a state s, we have to verify that for all schedulers $B \in Sched$ under consideration the value $prob_s^B(\{\pi \in Paths_{inf}^B(s) \ : \ \pi \models_{Sched} \phi\})$ is $\underline{\bowtie} \ p$.

Remark 6.
Note, that for a PNS and a $PCTL$ formula ϕ, we cannot decide wether $s \models_{Sched} [\phi]_{\trianglelefteq p}$ with $(\trianglelefteq \in \{\leq, <\})$ by just considering $PCTL$ formulae of the type $s \models_{Sched} [\phi']_{\trianglerighteq p}$ with $(\trianglerighteq \in \{\geq, >\})$. This was possible in the case of PDS (see equivalences 7 and 8 in section 2.3), but becomes impossible here, because the semantics quantifies over all schedulers.
In contrary to that, it suffices for $PCTL^*$ formulae to choose $\underline{\bowtie}$ from $\{\geq, >\}$ because of remark 4.

What kind of sets $Sched$ are we interested in ? There are two different sets of schedulers we will focus on, the set $Sched_{all}$ of all schedulers and the set $Sched_{fair}$ of fair schedulers, which will be defined in the following. First we give a definition of a fair path. For simplicity, we restrict our attention to strong fairness for all transitions. Other notions of fairness are considered in e.g. [50, 40, 42].

Definition 22. [Fairness of an infinite path]
Let $\mathcal{T} = (S, Steps, AP, L, \bar{s})$ be a PNS and π an infinite path of $\mathcal{T}$. π is called fair, iff the following holds:

$$\forall s \in lim(\pi) \ : \ \forall \mu \in Steps(s) \ : \ |\{j \ : \ \pi^j = s \ \wedge \ step(\pi, j) = \mu\}| = \infty$$

We denote the set of fair paths by $Path_{fair}$.[10]

[10] Note, that $Paths_{fair} \subseteq Paths_{inf}$.

That means a path π is fair, if and only if for each state s occuring infinitely often in π, each nondeterministic alternative in $Steps(s)$ is taken infinitely often in π (at s). For a scheduler B to be fair we require the set of fair paths to have measure 1.

Definition 23. [Fair Scheduler]
Let $\mathcal{T} = (S, Steps, AP, L, \bar{s})$ be a PNS and B be a scheduler for $\mathcal{T}$. We call B fair, if and only if $prob_s^B(Path_{inf}^B(s) \cap Path_{fair}) = 1$.

Note, that in the following we will write $\models_{all}$ instead of $\models_{Sched_{all}}$ and $\models_{fair}$, instead of $\models_{Sched_{fair}}$.
We now give an example of a PNS. We depict a PNS as follows. We use ellipses for the states. Thick arrows stand for the outgoing transitions from a state. A transition $\mu \in Steps(s) \backslash \{\mu_t^1 : t \in S\}$ is represented by a thick arrow that ends in a small circle, representing the probabilistic choice. We use directed thin arrows leading from the circle of a probabilistic choice to the possible successor states (i.e. all states t where $\mu(t) > 0$). A distribution $\mu_t^1 \in Steps(s)$ is represented by a thick arrow leading from s to t. The starting state has an extra incoming arrow and the state labelling is written in brackets.

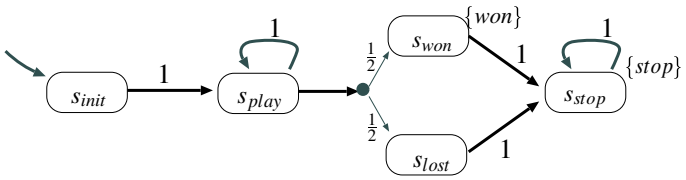


Fig. 6. The roulette player.

Figure 6 shows a simplified roulette player who keeps placing bets until his wife comes to the casino. That explains the nondeterministic choice in the state s_{play} (i.e. $Steps(s_{play}) = \{\mu^1_{s_{play}}, \mu'\}$, with $\mu'(s_{won}) = \mu'(s_{lost}) = \frac{1}{2}$). We have $L(s_{won}) = won$, $L(s_{lost}) = lost$ and $L(s') = \emptyset$ for the other states.
Observe that starting from state s_{init}, $\pi = s_{init}, s_{play}, s_{play}, s_{play}, \ldots$ is the only path that is not fair. So if the system starts in s_{init} and follows a fair scheduler, it will eventually reach the state s_{stop}. Considering the formula

$$\Phi \;=\; [\Diamond(won \wedge \mathcal{X} stop)]_{\geq 0.5}$$

it is easy to see that

$$s_{init} \not\models_{all} \Phi \qquad \text{but} \qquad s_{init} \models_{fair} \Phi.$$

3.2 Model Checking of *PCTL* against a *PNS*

Given a $PNS\ \mathcal{T} = (S, Steps, AP, L, \bar{s})$, a set $Sched$ of schedulers for $\mathcal{T}$ and a $PCTL$ state formula Φ, we want to decide, which states in S satisfy the formula Φ with respect to the schedulers under consideration. [11] In this section we present a model checking algorithm for this problem which is very similar to the one for PDS (see section 2.4). As in section 2.4, we first build the parse tree of the given formula Φ. We then process the parse tree bottom up, computing the satisfaction sets of the subformulae of Φ (i.e. $Sat(\Phi') = \{s \in S : s \models_{Sched} \Phi'\}$, where Φ' is a subformula of Φ).
As before we have $Sat(true) = S,\ Sat(a) = \{s \in S : L(s) = a\},\ Sat(\neg\Phi') = S \setminus Sat(\Phi')$ and $Sat(\Phi_1 \wedge \Phi_2) = Sat(\Phi_1) \cap Sat(\Phi_2)$ (independent of the choosen set $Sched$ of schedulers). So the only case left is $\Phi' = [\phi]_{\bowtie p}$, where ϕ is a $PCTL$ path formula, i.e. $\phi = \mathcal{X}\Phi'$ or $\phi = \Phi_1 \mathcal{U}^{\leq t} \Phi_2$ or $\phi = \Phi_1 \mathcal{U} \Phi_2$, with $t \in I\!N$. The cases concerning the next-step and bounded-until operator will be explained in the following section and how to deal with the unbounded-until operator will be explained afterwards.

Next-Step and Bounded-Until operator
We first describe the procedure for $s \models_{all} [\mathcal{X}\Phi']_{\underline{\bowtie} p}$ in detail. Following the semantics we have

$$s \models_{all} [\mathcal{X}\Phi']_{\underline{\bowtie} p} \Leftrightarrow prob_s^B(\{\pi \in Paths_{inf}^B(s) : \pi^1 \in Sat(\Phi')\}) \underline{\bowtie} p \ \forall B \in Sched_{all}.$$

For $B \in Sched_{all}$ we have

$$prob_s^B(\{\pi \in Paths_{inf}^B(s) \ : \ \pi^1 \in Sat(\Phi')\}) \ = \sum_{s' \in Sat(\Phi')} B(\pi^0)(s').$$

Since every possible distribution in $Steps(s)$ can be chosen by a scheduler and since we have to quantify over all schedulers, we get

$$s \models_{all} [\mathcal{X}\Phi']_{\trianglerighteq p} \quad \Leftrightarrow \quad \min_{\mu \in Steps(s)} \sum_{s' \in Sat(\Phi')} \mu(s') \trianglerighteq p,$$

where the latter can easily be checked algorithmically. Observe that the case $s \models_{all} [\mathcal{X}\Phi']_{\trianglelefteq p}$ is dealt with in the same way by replacing min by max.
Due to the fact that each mapping

$$B \ : \ \{\omega \in Paths_{fin} : |\omega| \leq t\} \ \longrightarrow \ \cup_{s \in S} Steps(s) \text{ with } B(\omega) \in Steps(last(\omega))$$

can be extended in a fair way, we get the same results for $s \models_{Sched} [\mathcal{X}\Phi']_{\underline{\bowtie} p}$ and $s \models_{Sched} [\Phi_1 \mathcal{U}^{\leq t} \Phi_2]_{\underline{\bowtie} p}$ independent of what $Sched$ was chosen to be. That leaves us with

[11] Remember, that we are only interested in $Sched \in \{Sched_{all}, Sched_{fair}\}$.

Lemma 4.

$$s \models_{Sched} [\mathcal{X}\Phi']_{\trianglerighteq p} \quad \Leftrightarrow \quad min_{\mu \in Steps(s)} \sum_{s' \in Sat(\Phi')} \mu(s') \trianglerighteq p$$

$$s \models_{Sched} [\mathcal{X}\Phi']_{\trianglelefteq p} \quad \Leftrightarrow \quad max_{\mu \in Steps(s)} \sum_{s' \in Sat(\Phi')} \mu(s') \trianglelefteq p$$

for $Sched \in \{Sched_{all}, Sched_{fair}\}$.

Having taken care of the case $[\mathcal{X}\Phi']_{\bowtie p}$, we now turn our attention to the case $[\Phi_1 \mathcal{U}^{\leq t} \Phi_2]_{\bowtie p}$ for which we give a recursive computation similar to the one in section 2.4 about PDS. The difference is, that this time we have to make sure to make the "worst" choice (of scheduler) in each step. The following lemma shows precisely what this means.

Lemma 5.
Let $Sched \in \{Sched_{all}, Sched_{fair}\}$. *For all* $s \in S$ *and* $j \in \mathbb{N}_0$ *we define* $p^{max}_{s,j}$ *and* $p^{min}_{s,j}$ *as follows:*

- *if* $s \in Sat(\Phi_2)$, *then* $p^{max}_{s,j} = p^{min}_{s,j} = 1 \quad \forall j \in \mathbb{N}_0$
- *if* $s \notin Sat(\Phi_1) \cup Sat(\Phi_2)$, *then* $p^{max}_{s,j} = p^{min}_{s,j} = 0 \quad \forall j \in \mathbb{N}_0$
- *if* $s \in Sat(\Phi_1) \setminus Sat(\Phi_2)$, *then* $p^{max}_{s,0} = p^{min}_{s,0} = 0$,

$$p^{min}_{s,j} \;=\; min_{\mu \in Steps(s)} \sum_{t \in S} \mu(t) \cdot p^{min}_{t,(j-1)},$$

$$p^{max}_{s,j} \;=\; max_{\mu \in Steps(s)} \sum_{t \in S} \mu(t) \cdot p^{max}_{t,(j-1)}.$$

Then, for all $s \in S$ *:*

$$s \models_{Sched} [\Phi_1 \mathcal{U}^{\leq t} \Phi_2]_{\trianglerighteq p} \;\; \textit{iff} \;\; p^{min}_{s,t} \trianglerighteq p \;\; \textit{and} \;\; s \models_{Sched} [\Phi_1 \mathcal{U}^{\leq t} \Phi_2]_{\trianglelefteq p} \;\; \textit{iff} \;\; p^{max}_{s,t} \trianglelefteq p$$

Proof :
It can be shown by induction on j, *that*

$$p^{extr}_{s,j} \;=\; extr_{B \in Sched} prob^B_s \{\pi \in Paths^B_{inf}(s) \;:\; \pi \models_{Sched} \Phi_1 \mathcal{U}^j \Phi_2\},$$

where $extr \in \{min, max\}$, *thus yielding the claim.*

The Unbounded-Until Operator
In contrary to the Next-Step and Bounded-Until operators, we have to distinguish different cases for the satisfaction relation of the Unbounded-Until operator according to the chosen set of schedulers.
So we first consider the case where the chosen set of schedulers is $Sched_{all}$. Given a state s we want to decide whether s satisfies $[\Phi_1 \mathcal{U} \Phi_2]_{\bowtie p}$,which is equivalent to

$$p^B_s(\Phi_1 \mathcal{U} \Phi_2) := prob^B_s(\{\pi \in Paths^B_{inf}(s) : \pi \models_{all} \Phi_1 \mathcal{U} \Phi_2\}) \bowtie p \quad \forall B \in Sched_{all}.$$

By the results of [10] we know that

$$sup\{p_s^B(\Phi_1 \mathcal{U} \Phi_2) : B \in Sched_{all}\} \;=\; max\{p_s^B(\Phi_1 \mathcal{U} \Phi_2) : B \in Sched_{simple}\} \text{ and}$$

$$inf\{p_s^B(\Phi_1 \mathcal{U} \Phi_2) : B \in Sched_{all}\} \;=\; min\{p_s^B(\Phi_1 \mathcal{U} \Phi_2) : B \in Sched_{simple}\}.^{12}$$

From now on we denote $max\{p_s^B(\Phi_1 \mathcal{U} \Phi_2) : B \in Sched_{simple}\}$ by $p_s^{max}(\Phi_1 \mathcal{U} \Phi_2)$ and $min\{p_s^B(\Phi_1 \mathcal{U} \Phi_2) : B \in Sched_{simple}\}$ by $p_s^{min}(\Phi_1 \mathcal{U} \Phi_2)$.
Since $Sched_{simple} \subseteq Sched_{all}$, we know that the supremum and infimum are really maximum and minimum. This fact gives rise to the following lemma.

Lemma 6.

$$\begin{array}{lllll} s \models_{all} [\Phi_1 \mathcal{U} \Phi_2]_{\trianglelefteq p} & \textit{iff} & p_s^{max}(\Phi_1 \mathcal{U} \Phi_2) \trianglelefteq p & \textit{iff} & s \models_{simple} [\Phi_1 \mathcal{U} \Phi_2]_{\trianglelefteq p} \\ s \models_{all} [\Phi_1 \mathcal{U} \Phi_2]_{\trianglerighteq p} & \textit{iff} & p_s^{min}(\Phi_1 \mathcal{U} \Phi_2) \trianglerighteq p & \textit{iff} & s \models_{simple} [\Phi_1 \mathcal{U} \Phi_2]_{\trianglerighteq p} \end{array}^{13}$$

We will describe later (see section 3.2) how to compute the values $p_s^{max}(\Phi_1 \mathcal{U} \Phi_2)$ and $p_s^{min}(\Phi_1 \mathcal{U} \Phi_2)$.
Now we consider the case where the chosen set of schedulers is $Sched_{fair}$. Given a state s, we want to decide, whether

$$prob_s^B(\{\pi \in Paths_{inf}^B(s) : \pi \models_{fair} \Phi_1 \mathcal{U} \Phi_2\}) \bowtie p \quad \forall B \in Sched_{fair}.$$

Here we have to deal with $\bowtie \in \{\geq, >\}$ and $\bowtie \in \{\leq, <\}$ in two different ways.

- $\bowtie \in \{\leq, <\}$: Since $p_s^{max}(\Phi_1 \mathcal{U} \Phi_2) = max\{p_s^B(\Phi_1 \mathcal{U} \Phi_2) : B \in Sched_{all}\}$ (see above), it obviously holds that

$$p_s^{max}(\Phi_1 \mathcal{U} \Phi_2) \;\geq\; p_s^F(\Phi_1 \mathcal{U} \Phi_2) \quad \forall F \in Sched_{fair}. \tag{1}$$

  Let now B be a scheduler such that $p_s^B(\Phi_1 \mathcal{U} \Phi_2) = p_s^{max}(\Phi_1 \mathcal{U} \Phi_2)$. It can be shown (see [4]) that there is a fair scheduler F_B with

$$p_s^{F_B}(\Phi_1 \mathcal{U} \Phi_2) \;\geq\; p_s^B(\Phi_1 \mathcal{U} \Phi_2) = p_s^{max}(\Phi_1 \mathcal{U} \Phi_2). \tag{2}$$

  From (1) and (2) it follows, that

$$\sup\{p_s^B(\Phi_1 \mathcal{U} \Phi_2) : B \in Sched_{fair}\} = \max\{p_s^B(\Phi_1 \mathcal{U} \Phi_2) : B \in Sched_{fair}\} =$$

$$p_s^{max}(\Phi_1 \mathcal{U} \Phi_2),$$

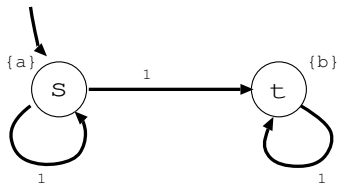
  which yields

  Lemma 7.

$$s \models_{fair} [\Phi_1 \mathcal{U} \Phi_2]_{\trianglelefteq p} \quad \textit{iff} \quad p_s^{max}(\Phi_1 \mathcal{U} \Phi_2) \trianglelefteq p.$$

[12] Note, that the set $Sched_{simple}$ is finite.
[13] This does not hold for $PCTL^*$ formulae. For a counterexample see section 4.

– $\bowtie \in \{\geq, >\}$: This case differs a little from the others. First, we give an example that shows that this case cannot be handled like the other cases. Consider the following $PNS\ \mathcal{T} = (\{s,t\}, Steps, \{a,b\}, L, s)$ with $Steps(s) = \{\mu_s^1, \mu_t^1\},\ Steps(t) = \{\mu_t^1\},\ L(s) = \{a\}$ and $L(t) = \{b\}$.



For the simple scheduler B with $B(s) = \mu_s^1$ (and $B(t) = \mu_t^1$) we get $p_s^B(a\mathcal{U}b) = 0$, thus $p_s^{min}(a\mathcal{U}b) = 0$. On the other hand we have $p_s^F(a\mathcal{U}b) = 1$ for each fair scheduler F and therefore $s \models_{fair} [a\mathcal{U}b]_{\geq 1}$. The problem in this example is, that the simple scheduler B with $B(s) = \mu_s^1$ forces the process to stay forever in the "non-successful" state s from which the "successful" state t can be reached. In contrary to this it can be shown for fair schedulers, that with probability 1 all states that are reachable from a state that is visited infinitely often, are also visited infinitely often. This explains, why $p_s^{min}(a\mathcal{U}b)$ cannot be "approximated" by fair schedulers. We now introduce some notation we will need to explain how to decide whether $s \models_{fair} [\Phi_1 \mathcal{U} \Phi_2]_{\trianglerighteq p}$.

Definition 24. [The sets $U_{>0}$ and $U_?$ and the corresponding atomic propositions]
Let $U_{>0}$ be the set of states for which there is a scheduler B, such that $p_s^B(\Phi_1 \mathcal{U} \Phi_2) > 0$, i.e.

$$U_{>0} \ = \ \{s \in S : p_s^{max}(\Phi_1 \mathcal{U} \Phi_2) > 0\}.$$

Let $U_? = U_{>0} \setminus Sat(\Phi_2)$.
Extend the set AP by two new atomic propositions $a_{>0}$ and $a_?$ and the labelling by
$a_{>0} \in L(s) \quad \Leftrightarrow \quad s \in U_{>0} \quad$ *and* $\quad a_? \in L(s) \quad \Leftrightarrow \quad s \in U_?$.

So $s \in U_{>0}$ if and only if starting from s it is possible to satisfy the formula $\Phi_1 \mathcal{U} \Phi_2$.
For each fair scheduler F and each state s, it can be shown (see [4]), that

$$prob_s^F(\{\pi \in Paths_{inf}^F(s) : \pi \models_{fair} \Phi_1 \mathcal{U} \Phi_2 \ \text{ or } \ \pi \models_{fair} true \mathcal{U} \neg a_{>0}\}) \ = \ 1.$$

That means, if the process follows the fair scheduler F, then with probability 1 the process will either satisfy the formula $\Phi_1 \mathcal{U} \Phi_2$ or will eventually reach a state that is not contained in $U_{>0}$. In other words, the set of infinite paths complying with the scheduler F and staying in $U_?$ forever, is a nullset (with respect to the measure defined by F). Since the set $\{\pi \in Paths_{inf}^F(s) : \pi \models_{fair} \Phi_1 \mathcal{U} \Phi_2 \ \text{ or } \ \pi \models_{fair} true \mathcal{U} \neg a_{>0}\}$ is the disjoint union of the two

sets $\{\pi \in Paths^F_{inf}(s) : \pi \models_{fair} \Phi_1 \mathcal{U} \Phi_2\}$ and $\{\pi \in Paths^F_{inf}(s) : \pi \models_{fair} a_? \mathcal{U} \neg a_{>0}\}$, it follows from the above, that

$$prob^F_s(\{\pi \in Paths^F_{inf}(s) : \pi \models_{fair} \Phi_1 \mathcal{U} \Phi_2\}) \;=$$

$$1 - prob^F_s(\{\pi \in Paths^F_{inf}(s) : \pi \models_{fair} a_? \mathcal{U} \neg a_{>0}\}).$$

So we get

$$s \models_{fair} [\Phi_1 \mathcal{U} \Phi_2]_{\trianglerighteq p} \quad \Leftrightarrow \quad p^F_s(\Phi_1 \mathcal{U} \Phi_2) \trianglerighteq p \;\; \forall F \in Sched_{fair} \quad \Leftrightarrow$$

$$p^F_s(a_? \mathcal{U} \neg a_{>0}) \trianglelefteq 1 - p \;\; \forall F \in Sched_{fair} \quad \Leftrightarrow \quad p^{max}_s(a_? \mathcal{U} \neg a_{>0}) \trianglelefteq 1 - p,$$

which leads to the following lemma

Lemma 8.

$$s \models_{fair} [\Phi_1 \mathcal{U} \Phi_2]_{\trianglerighteq p} \quad \textit{iff} \quad p^{max}_s(a_? \mathcal{U} \neg a_{>0}) \trianglelefteq 1 - p.$$

The only open question is how to efficiently compute the set $U_{>0}$, $U_?$ and the values $p^{max}_s(\Phi_1 \mathcal{U} \Phi_2)$ and $p^{min}_s(\Phi_1 \mathcal{U} \Phi_2)$, if the sets $Sat(\Phi_1)$ and $Sat(\Phi_2)$ have already be computed.

The Computation of $p^{max}_s(\Phi_1 \mathcal{U} \Phi_2)$, $p^{min}_s(\Phi_1 \mathcal{U} \Phi_2)$ and the set $U_{>0}$
Recall, that

$$U_{>0} = \{s \in S : p^{max}_s(\Phi_1 \mathcal{U} \Phi_2) > 0\}.$$

So we could compute $U_{>0}$, by computing the values $p^{max}_s(\Phi_1 \mathcal{U} \Phi_2)$ for all $s \in S$. But it is also easy to see, that $s \in U_{>0}$ if and only if starting from s, one can reach a state in $Sat(\Phi_2)$ via a path in $Sat(\Phi_1)$, i.e. $\exists\ \omega \in Paths_{fin}(s)$ such that $last(\omega) \in Sat(\Phi_2) \ \wedge \ \omega^i \in Sat(\Phi_1),\ i = 0, 1, \ldots, last(\omega) - 1$.
So a more efficient way for computing $U_{>0}$ is to perform a backwards reachability analysis from the states in $Sat(\Phi_2)$ within the states of $Sat(\Phi_1)$. Making this idea precise, we define the directed graph $G_{>0}(\Phi_1 \mathcal{U} \Phi_2) = (S, E_{>0})$, where

$$E_{>0} \;=\; \{(s,t) \in S \times S \mid t \in Sat(\Phi_1) \setminus Sat(\Phi_2) \quad \wedge \quad \exists\, \mu \in Steps(t) : \mu(s) > 0\}.$$

Then

$$U_{>0} \;=\; \{s \in S \mid s \text{ can be reached in } G_{>0}(\Phi_1 \mathcal{U} \Phi_2) \text{ from a state } s \in Sat(\Phi_2)\},$$

so $U_{>0}$ can be computed by a depth first search in $G_{>0}(\Phi_1 \mathcal{U} \Phi_2)$.
To compute $p^{max}_s(\Phi_1 \mathcal{U} \Phi_2), s \in S$, one can use linear programming techniques. It can be shown (see [10, 14]), that the values $p^{max}_s(\Phi_1 \mathcal{U} \Phi_2), s \in S$ are the unique solution of the linear minimization problem

$x_s = 1$ if $s \in Sat(\Phi_2)$
$x_s = 0$ if $s \notin U_{>0}$
$x_s \geq \sum_{t \in S} \mu(t) \cdot x_t$ if $s \in Sat(\Phi_1) \setminus Sat(\Phi_2), \quad \mu \in Steps(s)$,

where $\sum_{s \in S} x_s$ is minimized.
This problem can be solved by well known methods like the ellipsoid method or the simplex method. For $p_s^{min}(\Phi_1 \mathcal{U} \Phi_2)$ one gets an analogous linear maximization problem.
For reasons of completeness we also want to mention that one can use the characterization of the function $s \mapsto p_s^{max}(\Phi_1 \mathcal{U} \Phi_2)$ as the least fixed points of certain operators on certain function spaces and compute (resp. approximate) those values with iterative methods (see [5]).

The algorithm
Now we have everything on hand to present a model checking algorithm for $PCTL$ formulae against a PNS. As mentioned, one can build the parse tree of the given formula (or the parse graph to avoid multiple computations) and compute the satisfaction sets for the subformulae in a bottom up manner. In algorithm 3 and 4 we present a method that calculates the satisfaction sets recursively, so in an actual implementation one had to ensure that there were no multiple calls for syntactically identical subformulae.

Algorithm 3 PCTL_MODELCHECK

Input: PNS $\mathcal{T} = (S, Steps, AP, L, \bar{s})$, $PCTL$ state formula Φ over AP, $string \in \{all, fair\}$
Output: truth value of $\bar{s} \models_{string} \Phi$

IF $\bar{s} \in Sat(\Phi, string)$ **THEN**
    return *true*
ELSE
    return *false*

The Complexity of *PCTL* model checking against a *PNS*
In this paragraph we briefly estimate the time and space complexity of our model checking algorithm. Given a PNS $\mathcal{T}$ and a $PCTL$ state formula Φ we denote by n the number of states of $\mathcal{T}$, i.e. $n = |S|$ and by m the number of transitions of $\mathcal{T}$, i.e. $m = \sum_{s \in S} |Steps(s)|$. The number of subformulae of Φ is linear in $|\Phi|$.

- For a subformula Φ', where Φ' is *true*, an atomic proposition a or of the form $\neg\Phi''$ or $\Phi_1 \wedge \Phi_2$, the set $Sat(\Phi')$ can be computed in time $\mathcal{O}(n)$.
- If the subformula Φ' is of the form $[\mathcal{X}\Phi'']_{\bowtie p}$, the set $Sat(\Phi')$ can be computed in time $\mathcal{O}(m \cdot n)$, since for every $s \in S$ we have to compute the sum $\sum_{t \in Sat(\Phi'')} \mu(t)$ for every $\mu \in Steps(s)$.
- If the subformula Φ' is of the form $[\Phi_1 \mathcal{U}^{\leq t} \Phi_2]_{\bowtie p}$, the set $Sat(\Phi')$ can be computed in time $\mathcal{O}(t \cdot m \cdot n)$ (see algorithm).

Algorithm 4 $Sat(\Phi, string)$

Input:

Output: Set of all $s \in S$ that statisfy Φ with respect to the choosen set of schedulers

---

CASE Φ is

$true$: return S

a : return $\{s \in S : a \in L(s)\}$

$\Phi_1 \wedge \Phi_2$: return $Sat(\Phi_1, string) \cap Sat(\Phi_2, string)$

$\neg\Phi'$: return $S \setminus Sat(\Phi', string)$

$[\mathcal{X}\Phi']_{\trianglerighteq p}$: compute $Sat(\Phi', string)$;
$\forall s \in S:$ compute $p_{\mathcal{X}}^{min}(s) = \min_{\mu \in Steps(s)} \sum_{t \in Sat(\Phi', string)} \mu(t)$;
return $\{s \in S : p_{\mathcal{X}}^{min}(s) \trianglerighteq p\}$

$[\mathcal{X}\Phi']_{\trianglelefteq p}$: compute $Sat(\Phi', string)$;
compute $p_{\mathcal{X}}^{max}(s) = \max_{\mu \in Steps(s)} \sum_{t \in Sat(\Phi', string)} \mu(t)$;
return $\{s \in S : p_{\mathcal{X}}^{max}(s) \trianglelefteq p\}$

$[\Phi_1 \mathcal{U}^{\leq t} \Phi_2]_{\trianglerighteq p}$: compute $Sat(\Phi_1, string)$ and $Sat(\Phi_2, string)$;

$$\forall s \in S : p_0^{min}(s) = \begin{cases} 1 & \text{if } s \in Sat(\Phi_2, string) \\ 0 & \text{otherwise} \end{cases}$$

FOR $i = 1$ to t **DO**

$$\forall s \in S : p_i^{min}(s) = \begin{cases} 0, \text{ if } & s \notin Sat(\Phi_1, string) \cup Sat(\Phi_2, string), \\ 1, \text{ if } & s \in Sat(\Phi_2, string), \\ \min_{\mu \in Steps(s)} \sum_{t \in S} \mu(t) \cdot p_{i-1}^{min}(t) & \text{otherwise} \end{cases}$$

OD
return $\{s \in S : p_t^{min}(s) \trianglerighteq p\}$

$[\Phi_1 \mathcal{U}^{\leq t} \Phi_2]_{\trianglelefteq p}$: as above, replace "$\trianglerighteq$" by "$\trianglelefteq$" and "min" by "max"

$[\Phi_1 \mathcal{U} \Phi_2]_{\trianglelefteq p}$: compute $Sat(\Phi_1, string)$ and $Sat(\Phi_2, string)$;
compute $p_s^{max}(\Phi_1 \mathcal{U} \Phi_2)$ by solving a linear minimization problem;
return $\{s \in S : p_s^{max}(\Phi_1 \mathcal{U} \Phi_2) \trianglelefteq p\}$

$[\Phi_1 \mathcal{U} \Phi_2]_{\trianglerighteq p}$: compute $Sat(\Phi_1, string)$ and $Sat(\Phi_2, string)$;
IF $string$ = "all" **THEN**
compute $p_s^{min}(\Phi_1 \mathcal{U} \Phi_2)$;
return $\{s \in S : p_s^{min}(\Phi_1 \mathcal{U} \Phi_2) \trianglerighteq p\}$
ELSE (*$string$ = "fair"*)
compute $U_{>0}$, $U_?$ and the labelling for $a_{>0}$, $a_?$;
compute $p_s^{max}(a_? \mathcal{U} \neg a_{>0})$;
return $\{s \in S : p_s^{max}(a_? \mathcal{U} \neg a_{>0}) \trianglelefteq 1 - p\}$
ENDIF

END CASE

– If the subformula Φ' is of the form $[\Phi_1 \mathcal{U} \Phi_2]_{\bowtie p}$, the set $Sat(\Phi')$ can be computed in time $\mathcal{O}(poly(m,n))$. As mentioned above, $p_s^{max}(\Phi_1 \mathcal{U} \Phi_2)$, resp. $p_s^{min}(\Phi_1 \mathcal{U} \Phi_2)$ can be computed with well known methods (e.g. Ellipsoid-, Karmakar-method), which need polynomial time in the size of the problem. The needed graph analyses have time complexity $\mathcal{O}(m \cdot n)$.

Summing up over all subformulae of Φ, we get, that the model checking algorithms needs time

$$\mathcal{O}(|\Phi| \cdot (t_\Phi \cdot m \cdot n + p(m,n))),$$

where t_Φ stands for the maximal t in subformulae of the form $\Phi_1 \mathcal{U}^{\leq t} \Phi_2$[14] and $p(n,m)$ is a polynomial that represents the costs for solving the linear optimization problems.
For the estimation of the space complexity we assume w.l.o.g. that $n \leq m$. To represent the system $\mathcal{T}$ itself, we need space $\mathcal{O}(m \cdot n)$ (neglecting the representation for the labelling function L). For each subformula Φ', the set $Sat(\Phi')$ requires space $\mathcal{O}(n)$. The needed graph analyses require space $\mathcal{O}(m \cdot n)$ and for the computation of $p_s^{max}(\Phi_1 \mathcal{U} \Phi_2)$, resp. $p_s^{min}(\Phi_1 \mathcal{U} \Phi_2)$ space $\mathcal{O}(n^2)$ is needed.
Altogether this leads us to the following theorem:

Theorem 2.
Let $\mathcal{T}$ be a PNS and Φ a $PCTL$ state formula over the set of atomic propositions of $\mathcal{T}$. Then $Sat(\Phi)$ can be computed in time and space linear in the size of Φ and polynomial in the size of $\mathcal{T}$ if the set of schedulers under consideration is $Sched_{all}$ or $Sched_{fair}$.

Remark 7.
In [4] another classification of schedulers is considered, namely strictly fair schedulers. We call a scheduler B strictly fair if and only if all infinite paths that comply with B are fair. By $Sched_{sfair}$ we denote the set of strictly fair schedulers. For $\models_{Sched_{sfair}}$ some of the above calculations (the Unbounded-Until cases) are a little bit more complicated than for the fair schedulers (for details see [4]). However the results of theorem 2 still hold, if the set of schedulers under consideration is $Sched_{sfair}$.

3.3 Model Checking of $PCTL^*$ against a PNS

As in the deterministic case (see section 2.5) the model checking of PNS is extended to $PCTL^*$-formulae as well. In this section we explain how to model check a PNS against a $PCTL^*$-formula. Throughout this section let $\mathcal{T} = (S, Steps, AP, L, \bar{s})$ be a PNS. It follows from remark 4 that it is sufficient to consider formulae which do not make use of $[.]_{>p}$ and $[.]_{\geq p}$. The basic procedure is similar to the one in section 2.5. Let Φ be a $PCTL^*$ state formula. We only consider Φ to be of the form $\Phi = [\phi]_{\trianglelefteq p}$ since the other cases are either trivial or

[14] If there is no such subformula in Φ, then t_Φ is 1.

can be reduced to this one (see above). As in the deterministic case, we assume, that we have already computed (recursively) the satisfaction sets of all maximal state subformulae $\gamma_1, \ldots, \gamma_n$ of ϕ, so we can view them as atomic propositions. Then ϕ is a LTL formula over $\{\gamma_1, \ldots, \gamma_n\}$. [10] presents a model checking algorithm for $pCTL^*$ (which essentially coincides with our $PCTL^*$) that uses a normal form for the LTL formula ϕ. In [18, 17], an alternative method is considered, which uses the representation of LTL formulae by ω-automata and runs in time polynomial in the size of the system and doubly-exponential in the size of the formula (thus, this method meets the lower bound presented in [15]). In [10] and [18, 17] only the relation $\models_{all}$ is considered. We will first present briefly the model checking algorithm of [18, 17] and then explain, how it can be modified to handle fairness (see [4]).
The first step is the same for both cases ($\models_{all}$ and $\models_{fair}$). As in section 2.6, we construct a deterministic Rabin automaton $\mathcal{A} = (\Sigma, Q, \bar{q}, \rho, \alpha)$ such that $\mathcal{L}_\omega(\mathcal{A}) = \mathcal{L}_\omega(\phi)$. Recall that the size of the automaton $\mathcal{A}$ is doubly exponential in $|\phi|$ and that a run r of $\mathcal{A}$ is accepting if and only if there exists $(L, U) \in \alpha$, such that $lim(r) \cap L \neq \emptyset$ and $lim(r) \cap U = \emptyset$. After constructing the automaton $\mathcal{A}$, we build the product PNS $\mathcal{T}' = \mathcal{T} \times \mathcal{A} = (S \times Q, Steps', AP, L', (\bar{s}, \bar{q}))$, where

- $\mu' \in Steps'((s,q)) \iff$
  $\exists\ \mu \in Steps(s)$, s.t. $\mu'((t,p)) = \mu(t)$, if $p = \rho(q, L(s))$ and 0 otherwise.
- $L'((s,q)) = L(s)$.

Let α' be the lifted acceptance condition from $\mathcal{A}$, i.e.

$$\alpha' = \{(S \times L, S \times U) \mid (L, U) \in \alpha\}.$$

As in the deterministic case we get a one-to-one correspondence between the infinite paths of $\mathcal{T}$ and $\mathcal{T}'$ by path lifting which works as follows:

- First we embed S into $S \times Q$ by $s \mapsto (s, q_s)$, where $q_s = \rho(\bar{q}, L(s))$.
- Let $\pi = s_0 \xrightarrow{\mu_1} s_1 \xrightarrow{\mu_2} s_2 \ldots$ be an infinite path of $\mathcal{T}$. The induced infinite paths π' of $\mathcal{T}'$ is given by

$$\pi' = (s_0, p_0) \xrightarrow{\mu'_1} (s_1, p_1) \xrightarrow{\mu'_2} (s_2, p_2) \ldots,$$

  where $p_0 = q_s$, $p_{i+1} = \rho(p_i, L(s_i))$ and
  $\mu'_i((t,p)) = \begin{cases} \mu_i(t) & \text{, if } p = \rho(p_{i-1}, L(s_{i-1})) \\ 0 & \text{, otherwise.} \end{cases}$

The case $\models_{all}$
This case is very similar to the deterministic case, where accepting ergodic sets played an important role. Since now we have to quantify over all schedulers, we give the definition of a *maximal end component*, which is the pendant to an ergodic set in a markov chain.

Definition 25. [sub-PNS]
Given a PNS $\mathcal{T} = (S, Steps, AP, L, \bar{s})$*, a sub* $-$ *PNS of* $\mathcal{T}$ *is a tuple* $\mathcal{T}' = (S', Steps')$ *such that*

- $S' \subseteq S$ *and*
- $Steps'(s') \subseteq Steps(s') \quad \forall s' \in S'$.

Note, that a $sub - PNS$ $\mathcal{T}'$ of a PNS $\mathcal{T}$ (even when equipped with a state labelling and a starting state) is not a PNS, because $Steps'$ maps a state s' from $\mathcal{T}'$ to a probability distribution on the states of $\mathcal{T}$, so it can happen, that $supp(\mu) \supset S'$ for some $\mu \in Steps'(s')$ for some $s' \in S'$.

Definition 26. [End Component]
We call a $sub - PNS$ $\mathcal{T}' = (S', Steps')$ *an end component if :*

- $\forall s' \in S' \ : \ supp(\mu) \subseteq S' \quad \forall \mu \in Steps'(s')$.
- *The graph* $(S', E_{\mathcal{T}'})$ *is strongly connected,*
  where $E_{\mathcal{T}'} = \{(s', t') : s', t' \in S'$ *and* $\exists \mu \in Steps'(s') : \mu(t') > 0\}$.

Note that because of the first requirement, an end component is (accept for the lacking state labelling and the missing starting state) a PNS. The second requirement ensures, that for each pair (s', t') of states, there is always a scheduler, such that there is a finite path from s' to t'. It also holds, that given any starting state and a scheduler of a PNS, the induced process will end up with probability 1 in an end component (thus the name "end component")(see [18]).
We say that an end component $(S', Steps')$ is contained in another end component $(S'', Steps'')$ if

- $S' \subseteq S''$ and
- $Steps'(s') \subseteq Steps''(s') \quad \forall s' \in S'$.

Definition 27. [Maximal End Component]
An end component of a PNS $\mathcal{T}$ *is called maximal if it is not contained in any other end component of* $\mathcal{T}$.

As said before, maximal end components are the pendant to ergodic sets in markov chains. So the rest of the procedure is very similar to the deterministic case.
We call a maximal end component $(S', Steps')$ of $\mathcal{T}'$ accepting, if S' satisfies the acceptance condition α'. Let

$$Acc' = \{S' \mid (S', Steps') \text{ is an accepting maximal end component of } \mathcal{T}'\}.$$

Let acc be a new atomic proposition such that $acc \in L'(s')$ if and only if $s' \in Acc'$. One can then show (see [18]), that

$$sup_{B \in Sched_{all}} \ prob_s^B(\{\pi \mid \pi \in Paths_{inf}^B(s) : \pi \models_{all} \phi\}) \ =$$

$$sup_{B' \in Sched_{all}} \ prob_{(s,s_q)}^{B'}(\{\pi' \mid \pi' \in Paths_{inf}^{B'}((s, q_s)) : \pi' \models_{all} \Diamond acc\}),$$

where the latter quantity can be computed by the means of section 3.2. This leads to the following lemma.

Lemma 9.

$$s \models_{all} [\phi]_{\trianglelefteq p} \quad \textit{iff} \quad (s, q_s) \models_{all} [\Diamond acc]_{\trianglelefteq p}.$$

So we get

$$Sat(\Phi) \;=\; \{s \in S \mid p^{max}_{(s,q_s)}(\Diamond acc) \trianglelefteq p\}.$$

The case $\models_{fair}$
Not only do we have a one-to-one correspondence between the infinite paths of $\mathcal{T}$ and $\mathcal{T}'$ but it holds also, that the infinite path π of $\mathcal{T}$ is fair if and only if the corresponding path π' of $\mathcal{T}'$ is fair. We even get that each fair scheduler B of $\mathcal{T}$ induces a fair scheduler B' of T', such that $Paths^{B'}_{inf} = \{\pi' \mid \pi \in Paths^{B}_{inf}\}$. In addition, the associated probability spaces on $Paths^{B}_{inf}(s)$ and $Paths^{B'}_{inf}((s, q_s))$ are isomorphic.
For each $(L', U') \in \alpha'$, let $Acc'_{(L',U')}$ be the union of all subsets T' of $S \times Q$ such that $T' \cap U' = \emptyset$ and for all $t' \in T'$ the following hold:

- $supp(\mu') \subseteq T'$ for all $\mu' \in Steps(t')$
- $reach(t') \cap L' \neq \emptyset$.[15]

Let $Acc' = \cup_{(L',U') \in \alpha'} Acc'_{(L',U')}$ and let acc be an atomic proposition such that $acc \in L'(s')$ if and only if $s' \in Acc'$.
It then follows (see [4]) for all fair schedulers F' of $\mathcal{T}'$, that

$$prob^{F'}_{s'}(\Diamond acc) \;=\; prob^{F'}_{s'}(\{\pi' \in Paths^{F'}_{inf}(s') \;:\; L'(\pi'^0), L'(\pi'^1), \ldots \;\in \mathcal{L}_\omega(\mathcal{A})\}).$$

Therefore we get the following lemma.

Lemma 10.

$$s \models_{fair} [\phi]_{\trianglelefteq p} \quad \Longleftrightarrow \quad (s, q_s) \models_{fair} [\Diamond acc]_{\trianglelefteq p}.$$

So in order to compute the set $Sat(\Phi) = Sat([\phi]_{\trianglelefteq p})$, we have to compute the values $p^{max}_{(s,q_s)}(\Diamond acc)$ for $\mathcal{T}'$ by the means of section 3.2, i.e.

$$Sat(\Phi) \;=\; \{s \in S \mid p^{max}_{(s,q_s)}(\Diamond acc) \trianglelefteq p\}.$$

4 Concluding Remarks

In this paper we gave an overview of the logics $PCTL$ and $PCTL^*$ and model checking algorithms for deterministic and nondeterministic (finite-state) probabilistic systems. Here, we followed the state-labelled approach. Action-labelled variants of $PCTL/PCTL^*$ have been investigated as well, see e.g. [47, 45, 23, 9].

[15] Where $reach(t')$ denotes the set of states that are reachable from t', i.e. $reach(t') = \{s \in S \mid \exists\, \omega \in Paths_{fin} \;:\; t' = first(\omega) \text{ and } s = last(\omega)\}$.

In the remainder of this section we want to give a short overview on furthergoing topics.

In section 3.1 we defined schedulers for a PNS. There is a more general definition of schedulers, the so called stochastic schedulers. Given a probabilistic nondeterministic system $\mathcal{T}$, a stochastic scheduler of $\mathcal{T}$ is a function B mapping every finite path ω of $\mathcal{T}$ to a distribution $B(\omega) \in Distr(Steps(last(\omega)))$.
That means, after the process described by the PNS performed the first $|\omega|$ steps, the scheduler B chooses with the probability $B(\omega)(\mu)$ the next transition to be μ, for $\mu \in Steps(last(\omega))$. So if a system behaves according to a scheduler B and has followed the sequence $\omega = s_0, s_1, \ldots, s_n$ so far, then it will be in state t in the next step with probability

$$\sum_{\mu \in Steps(s_n)} B(\omega)(\mu) \cdot \mu(t).$$

Giving a stochastic scheduler B we can define a probability space in the straightforward way.
Furthermore, we can classify the stochastic schedulers into different classes (see [43]). A scheduler B is said to be of the kind

- (HS), if it is history-dependent and stochastic.
- (MS), if it is stochastic and markovian, that means $B(\omega) = B(last(\omega))$ for all finite path ω, i.e. B is history independent.
- (HD), if it is history-dependent and deterministic, that means for all finite path $\omega : B(\omega)(\mu) \in \{0,1\} \quad \forall\, \mu \in Steps(last(\omega))$, i.e. B schedules a unique next transition[16].
- (MD), if it is markovian and deterministic (also called simple).

Given a PNS, we will denote the set of all its stochastic schedulers by $Sched_{stoch}$. From now on we denote the set of schedulers of the kind (HD) by $Sched_{det}$, i.e. with the notations of section 3 $Sched_{det} = Sched_{all}$.
Given a PNS and a $PCTL$ path formula ϕ, we have already seen in section 3.2, that

$$s \models_{Sched_{det}} [\phi]_{\bowtie p} \quad \Leftrightarrow \quad s \models_{Sched_{simple}} [\phi]_{\bowtie p}$$

Moreover the following holds (see [10, 27, 46]):

- Given a PNS and a $PCTL$ path formula ϕ, then for all state s

$$s \models_{Sched_{stoch}} [\phi]_{\bowtie p} \quad \Leftrightarrow \quad s \models_{Sched_{simple}} [\phi]_{\bowtie p}.$$

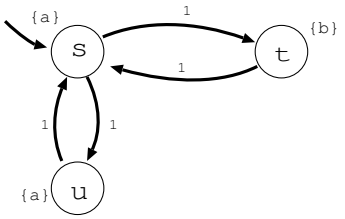
- Given a PNS and a $PCTL^*$ path formula ϕ, then for all state s

$$s \models_{Sched_{stoch}} [\phi]_{\bowtie p} \quad \Leftrightarrow \quad s \models_{Sched_{det}} [\phi]_{\bowtie p}.$$

[16] This coincides with definition 19.

An informal explanation for these results is the following: It can be shown (see [27, 46]), that the measure of a measurable set Ω w.r.t. a stochastic scheduler is a convex combination of measures of Ω w.r.t. deterministic schedulers. Since we are only interested in maximal and minimal measures, we get the above results. Hence, all methods presented in section 3 apply to stochastic schedulers as well.

Note, that for a $PCTL^*$ formula ϕ, it is not sufficient to consider only the simple schedulers as the following example demonstrates:



Consider the $PCTL^*$ formula $\phi = \Box a \vee \Box(a \rightarrow \mathcal{X} b)$.
Observe that due to the absence of probabilistic choice, starting in s, each scheduler will schedule a unique path. There are only two simple schedulers, the one that schedules the path $\pi = s, u, s, u, s, u, \ldots$. And the one that schedules the path $\pi' = s, t, s, t, s, t, \ldots$. Both paths satisfy the formula ϕ, so we get $s \models_{Sched_{simple}} [\phi]_{\geq 1}$. On the other hand, there is a scheduler in $Sched_{det}$, that schedules the path $\pi'' = s, t, s, u, s, t, s, u, \ldots$. This path does not satisfy the given formula ϕ and we get $s \models_{Sched_{det}} [\phi]_{\leq 0}$ and therefore $s \models_{Sched_{stoch}} [\phi]_{\leq 0}$.

As in the non probabilistic setting various simulation and bisimulation relations have been defined and studied in connection with probabilistic systems [32, 30, 46, 31]. It is possible to give logical characterizations for some of these relations, e.g. for $DTMCs$ bisimulation equivalence is the same as $PCTL^*$-equivalence (see e.g. [3]). Similar results for certain types of nondeterministic probabilistic systems and $PCTL^*$ have been established in [47, 29].

The state space explosion problem arises in the probabilistic setting to the same extent as in the non-probabilistic setting. In the last 10 years several methods have been developed to attack the state space explosion problem for non-probabilistic systems. Some of these approaches have been extended to the probabilistic case. In [16] a technique was introduced that involves an iterative usage of a simulation relation by starting with a small abstract model and successively refining it, until the properties under consideration are safely checked. Another approach are symbolic methods that use $MTBDDs$ (or other Decision Diagram variants) as an underlying data structure. Some of the model checking algorithms presented in this article were implemented (with several modifications) in the tool $PRISM$ ([37]) which uses standard methods of symbolic model checking (see e.g. [39, 7] and the article by D. Parker and A. Miner in this volume) and

in the tool *ProbVerus* [49].

A probabilistic variant of timed automata ([1]) and continous real time extensions of $PCTL$ were introduced and studied in [34–36]. See the article by Jeremy Sproston in this volume.

CSL, an extension of $PCTL$ for reasoning about performance measures and quantitative temporal properties of Continuos Time Markov Chains was introduced and studied in [3, 6]. The results were implemented in e.g. $E \vdash MC^2$ [28] and $PRISM$ [37].

References

1. Alur and Dill. A theory of timed automata. *Theoretical Computer Science*, pages 183–235, 1994.
2. Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking for probabilistic real-time systems (extended abstract). In Javier Leach Albert, Burkhard Monien, and Mario Rodríguez-Artalejo, editors, *Automata, Languages and Programming, 18th International Colloquium*, volume 510 of *Lecture Notes in Computer Science*, pages 115–126, Madrid, Spain, 8–12 July 1991. Springer-Verlag.
3. A. Aziz, V. Singhal, F. Balarin, and R. K. Brayton. It usually works: The temporal logic of stochastic systems. *Lecture Notes in Computer Science*, 939:155–165, 1995.
4. Baier and Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *DISTCOMP: Distributed Computing*, 11, 1998.
5. C. Baier. On algorithmic verification methods for probabilistic systems (habilitation thesis). November 1998.
6. C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time markov chains. *Lecture Notes in Computer Science, 10th International Conference on Concurrency Theory, CONCUR'99*, 1664:146–161, 1999.
7. Christel Baier, Edmund M. Clarke, Vassili Hartonas-Garmhausen, Marta Z. Kwiatkowska, and Mark Ryan. Symbolic model checking for probabilistic processes. In Pierpaolo Degano, Robert Gorrieri, and Alberto Marchetti-Spaccamela, editors, *Automata, Languages and Programming, 24th International Colloquium*, volume 1256 of *Lecture Notes in Computer Science*, pages 430–440, Bologna, Italy, 7–11 July 1997. Springer-Verlag.
8. Beauquier and Slissenko. Polytime model checking for timed probabilistic computation tree logic. *Acta Informatica 35*, pages 645–664, 1998.
9. Danièle Beauquier and Anatol Slissenko. Polytime model checking for timed probabilistic computation tree logic. *Acta Informatica*, 35(8):645–664, 1998.
10. A. Bianco and L. De Alfaro. Model checking of probabilistic and nondeterministic systems. *Lecture Notes in Computer Science*, 1026:499–513, 1995.
11. E. M. Clarke, E. Allen Emerson, and A. P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications: A practical approach. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
12. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 1999.

13. Costas Courcoubetis and Mihalis Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In *29th Annual Symposium on Foundations of Computer Science*, pages 338–345, White Plains, New York, 24–26 October 1988. IEEE.
14. Costas Courcoubetis and Mihalis Yannakakis. Markov decision processes and regular events (extended abstract). In Michael S. Paterson, editor, *Automata, Languages and Programming, 17th International Colloquium*, volume 443 of *Lecture Notes in Computer Science*, pages 336–349, Warwick University, England, 16–20 July 1990. Springer-Verlag.
15. Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, July 1995.
16. P.R. D'Argenio, B. Jeannet, H.E. Jensen, and K.G. Larsen. Reduction and refinement strategies for probabilistic analysis. In H. Hermanns and R. Segala, editors, *Proceedings of Process Algebra and Probabilistic Methods. Performance Modeling and Verification. Joint International Workshop, PAPM-PROBMIV 2001,* Copenhagen, Denmark, Lecture Notes in Computer Science. Springer-Verlag, 2001.
17. L. De Alfaro. Temporal logics for the specification of performance and reliability. *Lecture Notes in Computer Science*, 1200:165–??, 1997.
18. Luca de Alfaro. Formal verification of probabilistic systems. Thesi CS-TR-98-1601, Stanford University, Department of Computer Science, June 1998.
19. Emerson, Mok, Sistla, and Srinivasan. Quantitative temporal reasoning. In *Journal of Real Time System*, pages 331–352, 1992.
20. E. A. Emerson and J. Y. Halpern. 'Sometimes' and 'not never' revisited: on branching time versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.
21. E. Allen Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 995–1072. Elsevier Science Publishers, Amsterdam, The Netherlands, 1990.
22. W. Feller. *An Intro. to Probability Theory and its application.* John Wiley and Sons, New York, 1968.
23. H. Hansson. *Time and Probability in Formal Design of Distributed Systems.* Series in Real-Time Safety Critical Systems. Elsevier, 1994.
24. H. Hansson and B. Jonsson. A calculus for communicating systems with time and probabitilies. In IEEE Computer Society Press, editor, *Proceedings of the Real-Time Systems Symposium - 1990*, pages 278–287, Lake Buena Vista, Florida, USA, December 1990. IEEE Computer Society Press.
25. Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
26. Hans A. Hansson. Time and probability in formal design of distributed systems. SICS Dissertation Series 05, Swedish Institute of Computer Science, Box 1263, S-164 28 Kista, Sweden, 1994.
27. Sergiu Hart, Micha Sharir, and Amir Pnueli. Termination of probabilistic concurrent programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 5(3):356–380, July 1983.
28. H. Hermanns, J. P. Katoen, J. Meyer-Kayser, and M. Siegle. A markov chain model checker. *LNCS 1785*, pages 347–362, 2000.
29. R. Jagadeesan J. Desharnais, V. Gupta and P.Panangaden. Weak bisimulation is sound and complete for *pctl**. *Lecture Notes in Computer Science*, 2421:355–370, 2002.

30. Bengt Jonsson and Kim Guldstrand Larsen. Specification and refinement of probabilistic processes. In *Proceedings, Sixth Annual IEEE Symposium on Logic in Computer Science*, pages 266–277, Amsterdam, The Netherlands, 15–18 July 1991. IEEE Computer Society Press.
31. Chi-Chang Jou and Scott A. Smolka. Equivalences, congruences, and complete axiomatizations for probabilistic processes. In J. C. M. Baeten and J. W. Klop, editors, *CONCUR '90: Theories of Concurrency: Unification and Extension*, volume 458 of *Lecture Notes in Computer Science*, pages 367–383, Amsterdam, The Netherlands, 27–30August 1990. Springer-Verlag.
32. K. Larsen K and A. Skou. Bisimulation through probabilistic testing. *Inf Comput 94*, pages 1–28, 1991.
33. J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. Princeton University Press, 1960.
34. Marta Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Automatic verification of real-time systems with discrete probability distributions. *Lecture Notes in Computer Science*, 1601:75–95, 1999.
35. Marta Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Verifying quantitative properties of continuous probabilistic timed automata. *Lecture Notes in Computer Science*, 1877, 2000.
36. Marta Kwiatkowska, Gethin Norman, and Jeremy Sproston. Probabilistic model checking of the ieee 802.11 wireless local area network protocol. *Process Algebra and Probabilistic Methods, Performance Modeling and Verification, Second Joint International Workshop PAPM-PROBMIV 2002, Copenhagen, Denmark, July 25-26, 2002, Proceedings*, 2399:169–187, 2002.
37. Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM: Probabilistic symbolic model checker. In *Computer Performance Evaluation / TOOLS*, pages 200–204, 2002.
38. Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer, New York, 1992.
39. K. L. McMillan. *Symbolic Model Checking*. PhD thesis, Carnegie Mellon University, Pittsburgh, 1993.
40. A. Pnueli and L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1:53–72, 1986. They present a temporal logic for proving liveness properties of probabilistic concurrent programs based on the notion of "extreme fairness".
41. Amir Pnueli and Lenore Zuck. Probabilistic verification by tableaux. In *Proceedings, Symposium on Logic in Computer Science*, pages 322–331, Cambridge, Massachusetts, 16–18 June 1986. IEEE Computer Society.
42. Amir Pnueli and Lenore D. Zuck. Probabilistic verification. *Information and Computation*, 103(1):1–29, March 1993.
43. Martin L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994.
44. Shmuel Safra. On the complexity of ω-automata. In *29th Annual Symposium on Foundations of Computer Science*, pages 319–327, White Plains, New York, 24–26 October 1988. IEEE.
45. R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1995. Available as Technical Report MIT/LCS/TR-676.
46. R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Lecture Notes in Computer Science*, 836:481–496, 1994.

47. Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, Summer 1995.
48. Wolfgang Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 4, pages 133–191. Elsevier Science Publishers B. V., 1990.
49. S. Campos V. Hartonas-Garmhausen and E. Clarke. Probverus: Probabilistic symbolic model checking. *Lecture Notes in Computer Science*, 1601:96–110, 1999.
50. M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *26th Annual Symposium on Foundations of Computer Science*, pages 327–338, Los Angeles, Ca., USA, October 1985. IEEE Computer Society Press.
51. Moshe Y. Vardi. Probabilistic linear-time model checking: An overview of the automata-theoretic approach. *Lecture Notes in Computer Science*, 1601:265–276, 1999.
52. Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proceedings, Symposium on Logic in Computer Science*, pages 332–344, Cambridge, Massachusetts, 16–18 June 1986. IEEE Computer Society.