Automatic Verification of Probabilistic Systems

Tsuyoshi Morioka

Contents

1 Introduction

Temporal logics have been used to formally specify the behaviour of various computer systems, and *model checking* techniques have been successful in checking if finite-state systems satisfy a desirable property expressed as a temporal logic formula [8]. Since these systems are modeled as nondeterministic processes, a next logical step is to extend model checking techniques to probabilistic processes.

A system is called probabilistic if at any moment in time the system's next behaviour is decided according to a certain probability distribution over the possible actions at the time. In verification of probabilistic systems, the notion of correctness becomes also probabilistic; we can now ask the likelihood that a system exhibits a behaviour that satisfy a certain temporal property.

For example, consider a network system with a server and 20 clients, with the server that is capable of serving 10 requests at a time, and that drops any additional requests. We can model the entire system as a probabilistic system, assuming each client issues a request with a certain probability. Then a property such as "any request is eventually served" is false, because there is a nonzero probability that more than 10 requests are issued at the same time. But it is desirable to verify that any request will eventually be served *with probability*, say, 0.99. Or it is even better if we can verify that any request is served within 5 time steps with probability 0.9, or that the average time for the server to serve a request is 4.2 time steps. Note that the last two probabilistic properties explicitly express the performance characteristics of the system, which cannot be done with the traditional temporal logics. Similarly, the reliability property can be specified using probabilistic logics, e.g. "at any point in the execution of the system, deadlock occurs within 5 time steps with probability less than 0.01."

This survey is organized as follows. In Section 2, we introduce formalisms that are used to model probabilistic systems. In Section 3, we introduce various probabilistic temporal logics. We present the syntax and semantics of each logic, together with a discussion of model checking procedures. We conclude the paper by providing a big picture of current research in probabilistic model checking. Appendix A contains a summary of time complexities of model checking procedures in the logics we present in this paper.

2 Formal Models of Probabilistic Systems

In order to verify a probabilistic temporal property in a system, we need to construct a formal model of the system being verified.

Probabilistic systems are classified into two classes. A probabilistic system is called *purely probabilistic* if, for every state of the system, there is exactly one probability distribution over the next possible states; in other words, there is no nondeterminism in the system's behaviour.

A *generalized probabilistic system* is one that exhibits both probabilistic and nondeterministic behaviour. Typically nondeterminism arises from the asynchronity of concurrent processes, or the intervention from the environment over which the system has no control.

In this section we introduce several formalisms that are commonly used to model probabilistic systems.

2.1 Markov Chain

The *finite-state discrete-time Markov chain* is the most commonly used formalism for modelling purely probabilistic systems. Since we are concerned with discrete time and finite state systems, we use the term 'Markov chain' to refer to finite-state discrete-time Markov chain. Some researchers refer to them as 'labelled Markov chain.'

A Markov chain has a specified initial state, and for each state there is an assignment of truth values to atomic propositions. Every state is uniquely identified by the set of atomic propositions that are true in the state.

Definition 1 *(Markov chain) A Markov chain is a quadruple $M = (S, s_i, P, L)$, where*

- *S is a finite set of states,*
- *$s_i \in S$ is the initial state,*
- *$P : S \times S \to [0, 1]$ is a* transition probability function*, such that for all $s \in S$*
$$\Sigma_{t \in S} P(s, t) = 1,$$
- *$L : S \to 2^A$, where A is a set of atomic propositions, is a* labeling function*. L maps a state to a set of atomic propositions that are true in the state.* $\square$

Intuitively, the Markov chain represents a probabilistic system, which is in one of its states at all times. At each time unit, the system changes state according to a probability distribution given by the transition probability function P. Execution sequences arise by resolving the probabilistic choices. Formally, an *execution sequence* in M is an infinite sequence $\pi = s_0 s_1 \ldots$ where s_0 is the initial state of M, s_i are states and $P(s_i, s_{i+1}) > 0$ for $i = 0, 1, 2, \ldots$. A finite execution sequence can be extended to infinite ones by repeating its final state infinitely. Let $\pi(i)$ be the i-th state of π. Then the probability that M follows π is given by $P(\pi(0), \pi(1)) \times P(\pi(1), \pi(2)) \times P(\pi(2), \pi(3)) \times \ldots$. An execution sequence is also called a *path*.

Let $Path_M$ denote the set of all the paths which M can take. $Path_M(s)$ is the set of paths starting at s. If ω is a finite execution sequence starting at a state s, then $\Delta_M(\omega) = \{\pi \in Path_M(s) : \omega$ is a prefix of $\pi\}$. The probability measure *Prob* on the set $\Delta_M(\omega)$ of paths is $P(\omega) = P(s_0 s_1 \dots s_k) = P(s_0, s_1) \cdot P(s_1, s_2) \cdot \dots \cdot P(s_{k-1}, s_k)$.

2.2 Probabilistic Nondeterministic Systems

When we model a probabilistic system by a Markov chain, we must specify the transition probability of every possible state transition. However, in many practical settings it is often difficult to estimate the probabilities with which individual events occur. We may want to leave the probabilities unspecified, and model such transitions as nondeterministic transitions. Therefore it may be desirable to model a system by a formalism that allows both a probabilistic and a nondeterministic behaviour. For this purpose, Bianco and de Alfaro [7] introduced the *Probabilistic Nondeterministic System* (PNS).

Let S be the state space of a system. Then *next-state probability distribution* is a function $p : S \to [0, 1]$ such that $\Sigma_{s \in S} p(s) = 1$. Intuitively, $p(s)$ is the probability of making a direct transition to s from the current state.

Definition 2 *(PNS) A PNS* Π *is a quadruple* $\Pi = (S, s_i, L, \tau)$*, where*

- *S is the finite set of states;*
- *$s_i \in S$ is the initial state;*
- *$L : S \to 2^A$, where A is a set of atomic propositions, is a* labeling function*. L maps a state to a set of atomic propositions that are true in the state.*
- *τ is a function that associates with each $s \in S$ the set $\tau(s) = \{p_1^s, p_2^s, \dots, p_k^s\}$ of next-state probability distributions from s. We denote $|\tau(s)|$ by k_s.*
  □

A state transition of a PNS is made in a two-phase process. Let s be the current state. First, a next-state probability distribution p_j^s is selected nondeterministically among $\tau(s)$. Second, a successor state $t \in S$ is chosen according to the probability distribution p_j^s on S. It is clear that PNSs can model generalized probabilistic systems. For example, consider m purely probabilistic processes are running concurrently. If process i is modeled by a Markov chain A_i, then the entire system $A_1|A_2|\dots|A_m$ can be modeled as a PNS Π by assigning each state $s \in S$ a set $\{p_1^s, p_2^s, \ldots, p_m^s\}$ of m next-state probability distributions, where a distribution p_i^s arises from a move taken by the chain A_i. In this way, the probabilistic behaviour of each Markov chain is preserved, and the nondeterministic choices represent which process is scheduled to run.

Given a PNS Π, let a *reachability* relation $\rho \subseteq S \times S$ be defined by $\rho = \{(s,t) | \exists p^s \in \tau(s).p^s(t) > 0\}$. Informally, $\rho(s,t)$ holds iff there exists a nondeterministic choice that makes the probability of the transition from s to t nonzero. Then an *execution sequence*, or a *path* is defined as $\pi = s_1 s_2 \ldots$ such that $\rho(s_i, s_{i+1})$ for $i \geq 1$. $Path_\Pi$ is the set of all the paths which Π can take, and $Path_\Pi(s)$ is the set of paths starting at s. In the same way as in the case of Markov chains, if ω is a finite sequence of states, $\Delta_\Pi(\omega)$ denotes the set of all paths of which ω is their prefix.

Notice that, because of the presence of nondeterminism, we cannot define a probability measure of a set Δ_Π of paths. Instead, we define its *maximum probability* $Prob^+(\Delta_\Pi)$ and its *minimum probability* $Prob^-(\Delta_\Pi)$. Informally, $Prob^+(\Delta_\Pi)$ (resp. $Prob^-(\Delta_\Pi)$) is the probability that Π follows a sequence in Δ_Π provided the nondeterministic choices are as favorable (unfavorable) as possible.

Bianco and de Alfaro [7] introduces the notion of *strategies*. Some use the terms *schedular* [21, 3, 22] and *adversary* [4] to refer to the same thing. In this survey we prefer the term adversary. Informally, an adversary η is an agent external to the system which makes nondeterministic choices for the system. Hence at each step of an execution. the nondeterministic choice is resolved by an adversary, and once a nondeterministic choice is made, the system resolves the probabilistic choice. Therefore, given a PNS Π and an adversary η, it is possible to calculate the probability that the system follows a sequence in Δ_Π.

2.3 Timed Probabilistic Nondeterministic Systems

PNS can be effectively used to model a generalized probabilistic system in which every state transition takes the same amount of time. However, in many practical settings, different events take different amount of time. If that is the case, it is not easy to model such a system by a PNS. In that case, a *Timed Probabilistic Nondeterministic Systems* (TPNS) [11] can be used effectively.

TPNSs are similar to PNSs. To each state of a TPNS is associated a set of actions that can be chosen nondeterministically; the successor of the state is then determined by resolving the probabilistic choice arising from the action chosen. The major difference between TPNS and PNS is that in TPNS there is a cost is assciated with each choice of action The cost of an action is interpreted as the amount of time elapsed during the action. The cost of an action must be either 0 or 1.

Definition 3 *(TPNS) A TPNS Λ is defined as $\Lambda = (S, L, Acts, \kappa, p, c, s_i)$, where*

- *S is the finite set of states;*
- *$L : S \rightarrow 2^A$, where A is a set of atomic propositions, is a* labeling function*. L maps a state to a set of atomic propositions that are true in the state.*
- *$Acts$ is a finite set of actions.*
- *κ is a function that associates with each $s \in S$ the nonempty set $\kappa(s) \subseteq Acts$ of actions that can be taken at s.*
- *p is a probability distribution function such that, for all $s, t \in S$ and $a \in \kappa(s)$, $p(t|s, a)$ is the probability of a transition from s to t under action a. We require $\Sigma_{t \in S} p(t|s, a) = 1$ for all $s \in S$ and $a \in \kappa(s)$.*
- *c is a cost function such that, for all $s \in S$ and $a \in \kappa(s)$, $c(s, a) \in \{0, 1\}$ is the cost of performing the action a at s.*
- *$s_i \in S$ is the initial state;* $\Box$

Similarly to PNSs, a state transition of a TPNS is made in a two-phase process. Let s be the current state. First, an action $a \in \kappa(s)$ is chosen nondeterministically. Second, a successor state $t \in S$ is chosen according to the probability distribution $p(t|s, a)$. This process is repeated to give rise to the execution sequence of a TPNS.

An execution sequence (or a path) of a TPNS Λ is an infinite sequence $\pi = s_0 a_0 s_1 a_1 \ldots$ such that $s_i \in S$, $a_i \in \kappa(s_i)$ and $p(s_{i+1}|s_i, a_i) > 0$ for all $i \geq 0$. Let Δ_Λ be any set of paths of Λ, defined in a way similar to Δ_Π is defined. Since a TPNS exhibits a nondeterministic behaviour, the exact probability that Λ follows a path in Δ_Λ cannot be computed without specifying an adversary. Maximum probability $Prob^+(\Delta_\Lambda)$ and its minimum probability $Prob^-(\Delta_\Lambda)$ are defined in the same way as in the case of PNSs.

3 Probabilistic Logics

There have been significant developments in probabilistic model checking research over the last few years. In this section we are going to present a few probabilistic logics. For each logic, we present its syntax and semantics with respect to one of the formal models introduced above, and briefly discuss the model checking procedures.

3.1 PCTL*

Probabilistic real time Computational Tree Logic, PCTL, was developed by Hansson and Jonsson [14, 15] for expressing real-time and probability in systems. Baier, Kwiatkowska and Ryan [6] extended it to PCTL*, similarly to the way in which CTL* extends CTL. PCTL* is a much richer logic which allows more complex formulas. We will begin our discussion with PCTL*, and then introduce PCTL by restricting the syntax of PCTL*.

3.1.1 Syntax of PCTL*

Formulas in PCTL* are built from atomic propositions, propositional logical connectives and operators for expressing time and probabilities. There are two types of formulas in PCTL*: *state formulas* (which are true in a specific state) and *path formulas* (which are true along a specific path). The syntax of PCTL* is defined as follows.

Definition 4 *(PCTL* syntax) Let A be a finite set of atomic propositions. The state formulas of PCTL* are given by:*

$$\phi \quad := \quad tt \quad | \quad a \quad | \quad \phi_1 \wedge \phi_2 \quad | \quad \neg\phi \quad | \quad [f]_{\sqsupseteq p}$$

where $a \in A$ is an atomic proposition, p is a real value in $[0,1]$, $\sqsupseteq$ stands for either $>$ or $\geq$, and f is a path formula of PCTL:*

$$f \quad := \quad \phi \quad | \quad f_1 \wedge f_2 \quad | \quad \neg f \quad | \quad Xf \quad | \quad f_1 U^{\leq t} f_2$$

where t is an integer value. □

As in CTL*, X is a next-state operator and U is a *strong* Until. Note that the strong Until operator in PCTL* has a time bound t. Informally, an execution sequence of a Markov chain, PNS or TPNS satisfies $f_1 U^{\leq t} f_2$ if and only if f_2 becomes true within t time steps, and f_1 will be true between now and the time f_2 becomes true. When $t = \infty$, it is equivalent to the conventional Until operator in CTL*.

PCTL* differs from CTL* significantly in that the path quantifiers **A** and **E** are absent, and instead there is a probability operator $[f]_{\sqsupseteq p}$, where f is a path formula. The intuitive meaning of $[f]_{\geq p}$ is that with probability at least p the system follows a path that satisfies f (similarly for $[f]_{>p}$). Then it is clear that the probability operator generalizes the path quantifiers, since $\mathbf{A}f$ is equivalent to $[f]_{\geq 1}$, and $\mathbf{E}f$ to $[f]_{>0}$.

Using the Until operator, other temporal operators such as $F^{\leq t}$ ("eventually within t steps") and $G^{\leq t}$ ("always for the duration of t time steps") can be defined as $F^{\leq t}f = ttU^{\leq t}f$ and $G^{\leq t}f = \neg F^{\leq t}\neg f$, respectively. When $t = \infty$, we simply write F (G) instead of $F^{\leq t}$ $(G^{\leq t})$.

PCTL is a restricted subset of PCTL*, in which the probabilistic operator must be applied to every subformula of the form Xf or $f_1U^{\leq t}f_2$.

Definition 5 *(PCTL syntax) The state formulas of PCTL are defined in the same way as in PCTL*. The path formulas of PCTL are given by:*

$$f \quad := \quad X\phi_1 \quad | \quad \phi_1U^{\leq t}\phi_2$$

where ϕ_1, ϕ_2 are state formulas and t is an integer value. $\square$

Alternatively, the syntax of PCTL can be described as follows:

Definition 6 *(PCTL syntax) Let A be a set of atomic propositions. A PCTL formula ϕ is defined inductively as follows:*

$$\phi \quad := \quad tt \quad | \quad a \quad | \quad \phi_1 \wedge \phi_2 \quad | \quad \neg\phi \quad | \quad [X\phi]_{\sqsupseteq p} \quad | \quad [\phi_1U^{\leq t}\phi_2]_{\sqsupseteq p}$$

where $a \in A$ is an atomic proposition, p is a real value in $[0,1]$, $\sqsupseteq$ stands for either $>$ or $\geq$. $\square$

The definitions 5 and 6 are equivalent.

To illustrate the difference in the expressive powers of PCTL and PCTL*, consider formulas $[FGa]_{\geq 0.7}$ ("with probability at least 0.7 eventually a holds forever") and $[Fa \wedge Fb]_{\geq 0.1}$ ("with a probability of at least 0.1, both a and b eventually holds.") Both formulas are PCTL* formulas but neither can be expressed in PCTL. The reason is that F^{leqt} and $G^{\leq t}$ are defined from the U operator, and in PCTL the probabilistic operator must be applied to every U operators in a formula.

3.1.2 PCTL* and Markov Chains

Originally, Hansson and Jonsson [14, 15] defined the semantics of PCTL with respect to Markov chains. It is easy to see that the semantics of PCTL* w.r.t. Markov chains is defined in the same way.

Definition 7 *(PCTL* semantics w.r.t. a Markov chain) Let M be a Markov chain, s a state of M, and π a path. Given a PCTL* formula f, the relation $\models$ is inductively defined as follows:*

- $M, s \models a$ *iff* $a \in L(s)$
- $M, s \models \neg f$ *iff not* $s \models f$
- $M, s \models f \wedge g$ *iff* $s \models f$ *and* $s \models g$
- $M, \pi \models fU^{\leq t}g$ *iff there exists an* $i \leq t$ *such that* $\pi(i) \models g$ *and* $\pi(j) \models f$ *for all* $0 \leq j < i$
- $M, \pi \models Xf$ *iff* $\pi(1) \models f$
- $M, s \models [f]_{\sqsupseteq p}g$ *iff* $Prob\{\pi \in Path(s) : \pi \models f\} \sqsupseteq p$

□

It is important to note that, when a PCTL* formula is interpreted w.r.t. a Markov chain M, one state transition of M is interpreted as one time step. If we are modeling a system in which different events take different amounts of time, Markov chains and PCTL* is an awkward framework to work with. This problem is addressed by the logic pTL* and TPNS in Section 3.3.

Hansson and Jonsson ([14, 15]) gave the first model checking algorithm which decides if a PCTL formula is satisfied by a given Markov chain. It was based on the CTL model checking algorithm. Here we give a high-level description of the algorithm. First, it decomposes the formula into the smallest subformulas, and labels the states in which each subformula is satisfied. Then it inductively labels the states where a bigger subformula is satisfied, until it finds states in which the given formula holds.

The time complexity of their algorithm is $O(|S|^3 \times |\phi|)$ where S is the set of states in M and ϕ is a PCTL formula. Although this is a polynomial time algorithm, it involves building a transition probability matrix and perform expensive matrix multiplication operations, and hence it cannot be useful in practical settings. Recently, symbolic techniques for the PCTL model checking have been developed. See [6, 16].

It has been shown [2] that the model checking in PCTL* w.r.t. Markov chains can be done in time polynomial in the size of the system. However, in terms of the size of the formula, the time complexity is single exponential.

3.1.3 PCTL* and PNS

PCTL* formulas can be interpreted w.r.t. Probabilistic Nondeterministic Systems (PNS) [7]. Since it is no longer possible to calculate the probability measure of a set of paths, the semantics of PCTL* w.r.t. Markov chains must be modified to fit PNSs:

Definition 8 *(PCTL* semantics w.r.t. a PNS) Let* Π *be a PNS and* s *a state of* M*. The satisfaction relation* $\models$ *is defined in the same way as in Definition 6, with the following modification:*

- $M,s \models [f]_{\sqsupseteq p} g$ *iff* $Prob^{+}\{\pi \in Path(s) : \pi \models f\} \sqsupseteq p$

$\square$

Informally, a PCTL* formula of the form $[\phi]_{\geq p}$ is satisfied by a PNS Π if and only if there is probability at least p that Π follows a path that satisfies ϕ *under every possible adversary.*

Bianco and de Alfaro gave model checking algorithms [7] for PCTL and PCTL*. The model checking for PCTL and PCTL* on PNSs can be done in the time polynomial in the size of the PNS. However, in terms of the size of the formula, single exponential time is required for the PCTL model checking, and doubly exponential time for the PCTL* model checking.

3.2 PBTL

Recall that a PCTL* formula is satisfied by a PNS Π if and only if Π satisfies it under every possible adversary. This definition may not be satisfactory, since there may be an adversary η that makes many liveness properties trivially false by, say, scheduling one process infinitely often and letting other processes starve. Consider, as an example, a producer process and a consumer process in a concurrent system. A property $[G(item\_produced \rightarrow F(item\_consumed))_{\geq 0.9}$ can be made trivially false, since there may exist an adversary (or a scheduler) that never schedules the consumer to run. Hence, we may want to exclude such "unfair" adversaries from our consideration when we talk about the correctness proof of a formula. And there is a good reason to do so: in many practical situations, such unfair adversaries are not supposed to be used.

Therefore, it is desirable to decide if a given formula of a probabilistic temporal logic is satisfied *provided some fairness constraint is being satisfied.* That is exactly what Baier and Kwiatkowska did [3, 4, 5]. They introduced a new logic PBTL (which stands for Probabilistic Branching Time Logic), and developed algorithms to check whether a given PBTL formula is satisfied by a PNS under "fair" adversaries.

PBTL is an extension of PCTL. Syntactically, PBTL is almost identical to PCTL. Essentially, what Baier and Kwiatkowska did was augment PCTL with universal and existential quantifiers *over adversaries* for X and U operators. Informally, a PBTL formula $[f_1 \forall U^{\leq t} f_2]_{\sqsupseteq p}$ expresses that under every adversary $[f_1 U^{\leq t} f_2]_{\sqsupseteq p}$ is true. If we replace $\forall$ by $\exists$, then the resulting formula says that there exists a adversary that gives rise to sequences in which the formula is true. Quantifiers for the X operator are defined in a similar way.

Although Baier and Kwiatkowska defines a new formal model of their own and define the PBTL semantics w.r.t. it, it is a variant of PNS [4]. Therefore we discuss the PBTL semantics w.r.t. PNSs. The satisfaction relation for PBTL is defined as follows. Given a PNS Π, a PBTL formula ϕ, and a set V of adversaries, the satisfaction relation $\models_V$ holds iff every execution sequence of Π under any adversary in V satisfies ϕ. When V is a set of fair adversaries, the satisfaction relation is denoted as $\models_{fair}$.

Fairness of an adversary is defined w.r.t. nondeterministic choices. Let s be a state of a PNS Π, and $\tau(s)$ be the set of next-state probability distributions at s. Then a sequence π is said to fair iff every next-state probability distribution $p_i^s \in \tau(s)$ is taken by an adversary η infinitely often, provided s appears in π infinitely often. An adversary η is said to be fair iff the measure of its fair execution sequences is 1.

Baier and Kwiatkowska proved a surprising result: in order to prove that a given PBTL formula holds in all fair sequences, we only need to investigate a certain finite subclass of (in general unfair) sequences [3, 4]. Based on this result they gave a model checking algorithm for PBTL on PNS. Their algorithm is based on Baier and de Alfaro's algorithm for PCTL model checking on PNS [7], and has the same time complexity (polynomial in the size of the system and linear in the size of the formula).

Recently, results in investigation of symbolic method of PBTL model checking came out. Kwiatkowska et. al. [18] and de Alfaro et. al. [12] developed symbolic model checkers for PBTL using the Multi-Terminal Binary Decision Diagrams, which extends the Binary Decision Diagrams. They used an existing MTBDD package (CUDD, the Colorado University Decision Diagrams) and implemented simplex methods in terms of MTBDDs. De Alfaro et. al. [12] improved it. Their tools allowed them to model check purely probabilistic systems of up to 10^{25} states, and generalized probabilistic systems of up to 10^{30} states. Also, they found that model checking of qualitative properties (expressed as 'probability 1') was very fast. Their tools can be used to perform the PCTL model checking, since PCTL is a subset of PBTL.

PBTL can be extended in a natural way to PBTL*. The survey author believes that the PBTL* model checking has the same time complexity as the PCTL* case. However, no results on PBTL* have been known to the author.

3.3 pTL*

In specifications written in the logics we've discussed so far, one time unit is interpreted as one transition in the model of the system. When modeling real-time systems, this may not be desirable in general, since it is possible for different events in systems to take different amounts of time. For example, in a communication system over a long distance network, the time it takes to deliver a message from a sender to a receiver may depend on how far the two parties are and the amount of data being sent. It may be possible to model such a system as a PNS, but definitely it is not convenient to do so.

The logic pTL*, together with TPNSs (Timed Probabilistic Nondeterministic Systems), are developed to address this issue. The logic pTL* is introduced by de Alfaro [10, 11]. (He never states what pTL* stands for.) His motivation is to extend the range of applications of probabilistic temporal logics to include the study of performance and reliability properties of systems. In pTL*, the timing properties are expressed using *instrumentation clocks* (see below for the definition of instrumentation clocks). pTL* is an extension of CTL*, with two operators added: the operator **P** is used to express bounds on the probability of system behaviours, and the operator **D** is used to express bounds on the *average time between events.*

3.3.1 Syntax of pTL*

In order to define pTL* syntax and semantics, we need to introduce the instrumentation clocks.

Definition 9 *(Instrumentation Clocks) An* instrumentation clock δ *over a state space* S *is a pair* (δ_T, δ_S)*, where:*

- $\delta_\tau \subseteq S \times S$ *is a set of transitions in which the clock* δ *is reset, and*

- $\delta_S \subseteq S$ *is a set of states in which the clock is advanced.*

Thus, δ *measures the number of times the system has visited the states in* δ_S *since a state in* δ_τ *has been visited last time.* □

Definition 10 *(The pTL* Syntax) There are two types of pTL* formulas: the state formulas and the path formulas. Given a set* A *of atomic propositions and a set* C *of instrumentation clocks, the state formulas are give by:*

$$\begin{array}{rcl} \phi & := & tt \quad | \quad a \quad | \quad \phi_1 \wedge \phi_2 \quad | \quad \neg\phi \quad | \quad P_{\bowtie b} f \quad | \\ & & D_{\bowtie d}\phi \quad | \quad c > k \quad | \quad c = k \quad | Af \quad | \quad Ef \end{array}$$

where $a \in A$ *is an atomic proposition,* $c \in C$ *is an instrumentation clock,* $k \in \mathbb{N}$, $b \in [0,1]$, $d \geq 0$, $\bowtie$ *stands for one of* $[<, \leq, >, \geq]$. A *and* E *are the path quantifiers taken from CTL, and* f *is a path formula give by:*

$$f \quad := \quad \phi \quad | \quad f_1 \wedge f_2 \quad | \quad \neg f \quad | \quad f_1 U f_2$$

□

The survey author believes that the path quantifiers are present just to make formulas more readable; Af can be written as $P_{\geq 1} f$, and Ef as $P_{>0} f$. The temporal operators F and G can be defined in a usual way using the U operator.

Let us give a couple examples of pTL* formulas. The formula $AG[sendmessage \rightarrow P_{\geq 0.9} F(receivemessage)]$ specifies that whenever there is an attempt to send a message, there is a probability of at least 0.9 that it will be eventually delivered. Similarly, the formula $AG[sendmessage \rightarrow D_{<5} receivemessage]$ specifies that whenever a message is sent, it will be received within 5 time steps, on average.

pTL* is richer than PCTL* in that it can specify properties expressible in PCTL*, and other properties, such as ones that express the average time between events.

3.3.2 TPNS and pTL*

The semantics of a pTL* formula is defined w.r.t. a TPNS Π and a set of instrumentation clocks C. We do not wish to give the definition here, since it involves the construction of an *instrumented* TPNS Π_C^ϕ whose states keep track of the value of the clocks. The satisfaction of ϕ is defined on Π_C^ϕ.

De Alfaro [10, 11] showed that the pTL* model checking w.r.t. TPNSs can be done in time polynomial in the size of Π and doubly exponential in the size of the formula.

3.4 Qualitative Approach with LTL

The first probabilistic model checking algorithm was for checking if an LTL formula is satisfied in a system modeled by a Markov chain *with probability 1*. This approach is called *qualitative*, for an obvious reason. Vardi [21, 22] first developed a procedure to decide if an LTL formula ϕ holds in a given Markov chain M with probability 1. His algorithm runs in the time polynomial of the size of M but doubly exponential of the size of ϕ. Courcoubetis and Yannakakis [9] improved the results by showing that the time complexity of the problem is exponential in the size of ϕ and linear in the size of M. They also showed that the LTL model checking w.r.t. PNSs is complete for doubly exponential time in the size of the formula, and can be solved in time doubly exponential in the size of the formula and quadratic in the size of the system.

Baier and Kwiatkowska [4] showed that, in order to show a linear-time property is satisfied in a system with probability 1, it suffices to prove that all *fair* execution sequences satisfy the property, for some instance of general notion of fairness.

Iyer and Narashimha successfully applied the qualitative technique of LTL verification to the probabilistic version of Lossy Channel Systems, which has been shown to be undecidable in general case [17].

4 Conclusion

We have presented a few of the recent developments in model checking of probabilistic systems. We saw how probabilistic temporal logics can express the relative likelihood of temporal properties being satisfied in a given system, and how they can be used to specify the performance and reliability properties. We presented five logics, PCTL*, PCTL, PBTL, and pTL*. The first two are rather straightforward extensions of traditional temporal logics, and the rest are further extensions of the first two.

One of the probabilistic logics we did not discuss is pCTL*, which was introduced by Aziz et. al. [2]. It is almost identical to PCTL* except some minor differences, and all the results in PCTL* also apply to pCTL*. Naturally, a more restricted logic pCTL is obtained from pCTL*, in the same way PCTL is obtained from PCTL*. PTCTL [19, 20] and TPCTL [13] are two other logics we did not discuss in this paper.

Among a variety of modeling frameworks proposed, we presented two of the most popular ones: the Markov chains and the Probabilistic-Nondeterministic Systems. In order to discuss the logic pTL*, we also introduced the Timed Probabilistic Nondeterministic Systems. There are other frameworks such as Stochastic Real-Time Systems [10], Probabilistic Timed Graph and Probabilistic Timed Structure, but we did not discuss them in detail. The last two are defined to provide a semantics of PTCTL [19, 20].

The experience report by Alur et.al. [1] is the only published case study of probabilistic model checking in an industrial setting, and in our opinion it illuminate the usefulness of probabilistic model checking. They used a symbolic model checker TPWB for a logic TPCTL. However, the details about TPWB and TPCTL are unknown to the author at the time of writing. Interested readers are referred to [13].

In order to successfully verify large-scale probabilistic systems, we need efficient model checkers. Research in symbolic techniques in probabilistic model checking seems very promising in this regard. Hartonas-Garmhausen, Campos and Clarke [16] implemented a symbolic model checker ProbVerus, for PCTL verification on purely probabilistic systems. It has already been used to verify an industrial system with 10^{27} states. Other researchers [18, 12] developed experimental tools for PBTL symbolic model checking.

References

[1] R. Alur, L. J. Jagadeesan, J. J. Kott, and J. E. V. Olnhausen. Model-checking of real-time systems: A telecommunications application. In *Proceedings of the 19th International Conference on Software Engineering (ICSE '97)*, pages 514–524, Berlin - Heidelberg - New York, May 1997. Springer.

[2] 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.

[3] C. Baier and M. Z. Kwiatkowska. Automatic verification of liveness properties of randomized systems. In *Proceedings of the Sixteenth An-*

nual ACM Symposium on Principles of Distributed Computing, page 295, Santa Barbara, California, 21–24 Aug. 1997.

[4] C. Baier and M. Z. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *DISTCOMP: Distributed Computing*, 11, 1998.

[5] C. Baier and M. Z. Kwiatkowska. On the verification of qualitative properties of probabilistic processes under fairness constraints. *Information Processing Letters*, 66(2):71–79, 1998.

[6] C. Baier, M. Z. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. Technical Report CSR-97-2, University of Birmingham, School of Computer Science, Jan. 1997.

[7] A. Bianco and L. De Alfaro. Model checking of probabilistic and nondeterministic systems. *Lecture Notes in Computer Science*, 1026:499–513, 1995.

[8] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[9] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, July 1995.

[10] L. de Alfaro. Formal verification of performance and reliability of real-time systems. Technical Report CS-TR-96-1571, Stanford University, 1996.

[11] L. de Alfaro. Temporal logics for the specification of performance and reliability. *Lecture Notes in Computer Science*, 1200:165–176, 1997.

[12] L. de Alfaro, M. Z. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of probabilistic processes using MTBDDs and the Kronecker representation. Technical Report CSR-99-11, University of Birmingham, 1999.

[13] L. Fredlund. The timing and probability workbench: A tool for analyzing timed processes. Technical report, Uppsala University, Department of Computer Systems, 1994.

[14] H. Hansson and B. Jonsson. A framework for reasoning about time and reliability. In I. C. S. Press, editor, *Proceedings of the Real-Time Systems Symposium - 1989*, pages 102–111, Santa Monica, California, USA, Dec. 1989. IEEE Computer Society Press.

[15] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[16] V. Hartonas-Garmhausen, S. Campos, and E. Clarke. ProbVerus: Probabilistic symbolic model checking. *Lecture Notes in Computer Science*, 1601:96–110, 1999.

[17] P. Iyer and M. Narasimha. Model checking probabilistic lossy channel systems. Technical Report TR-98-08, Department of Computer Science, North Carolina State University, June 8 1998.

[18] M. Z. Kwiatkowska, G. Norman, D. A. Parker, and R. Segala. Symbolic model checking of concurrent probabilistic systems using MTBDDs and simplex. Technical Report CSR-99-1, University of Birmingham, School of Computer Science, Jan. 1999.

[19] M. Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. Technical Report CSR-98-10, University of Birmingham, School of Computer Science, Dec. 1998.

[20] M. Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Verifying quantitative properties of continuous probabilistic real-time graphs. Technical Report CSR-98-11, University of Birmingham, School of Computer Science, Dec. 1998.

[21] M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *focs85*, pages 327–338, 1985.

[22] M. Y. Vardi. Probabilistic linear-time model checking: An overview of the automata-theoretic approach. In *ARTS: International AMAST Workshop on Formal Methods for Real-Time and Probabilistic Systems (ARTS)*. LNCS, 1999.

A Summary of Time Complexity

The time complexities of the model checking procedures are summarized in Table 1. A system is said to be purely probabilistic if its behaviour can be completely specified by a probabilistic distribution. A system is called a generalized probabilistic system if it exhibits both a probabilistic behaviour and a nondeterministic behaviour. (These terms are defined in the next section.) S stands for the size of a system, and F for the size of a formula.

Logic	Purely Probabilistic Systems	Generalized Probabilistic Systems
LTL	poly(S) and exp(F)	poly(S) and doubly-exp(F) (2EXPTIME-complete in F)
PCTL	poly(S) and linear(F)	poly(S) and linear(F)
PCTL*	poly(S) and exp(F)	poly(S) and doubly-exp(F)
PBTL	N/A	poly(S) and linear(F)
pTL*	N/A	poly(S) and doubly-exp(F)

Table 1: Time complexities of model checking procedures in probabilistic logics. S denotes the size of the system being verified and F stands for the size of a formula that specify a probabilistic temporal property. $Poly(x)$, $linear(x)$, $exp(x)$ and $doubly - exp(x)$ stand for polynomial in x, linear in x, exponential in x, and doubly-exponential in x, respectively.