Statistical Model Checking of Black-Box Probabilistic Systems

Koushik Sen, Mahesh Viswanathan, Gul Agha
Department of Computer Science,
University of Illinois at Urbana-Champaign.
{ksen,vmahesh,agha}@uiuc.edu

Abstract. We propose a new statistical approach to analyzing stochastic systems against specifications given in a sublogic of continuous stochastic logic (CSL). Unlike past numerical and statistical analysis methods, we assume that the system under investigation is an *unknown, deployed black-box* that can be passively observed to obtain sample traces, but cannot be controlled. Given a set of executions (obtained by Monte Carlo simulation) and a property, our algorithm checks, based on statistical hypothesis testing, whether the sample provides evidence to conclude the satisfaction or violation of a property, and computes a quantitative measure (p-value of the tests) of confidence in its answer; if the sample does not provide statistical evidence to conclude the satisfaction or violation of the property, the algorithm may respond with a "don't know" answer. We implemented our algorithm in a Java-based prototype tool called VeStA, and experimented with the tool using case studies analyzed in [16]. Our empirical results show that our approach may, at least in some cases, be faster than previous analysis methods.

1 Introduction

Stochastic models and temporal logics such as continuous stochastic logic (CSL) [1, 3] are widely used to model practical systems and analyze their performance and reliability. There are two primary approaches to analyzing the stochastic behavior of such systems: *numerical* and *statistical*. In the numerical approach, the formal model of the system is *model checked* for correctness with respect to the specification using symbolic and numerical methods. Model checkers for different classes of stochastic processes and specification logics have been developed [8, 14, 13, 4, 5, 2, 6]. Although the numerical approach is highly accurate, it suffers from being computation intensive. An alternate method, proposed by Younes and Simmons [17], is based on Monte Carlo simulation and sequential hypothesis testing. Being statistical in nature, this approach is less accurate and only provides probabilistic guarantees of correctness. The approach does not assume knowledge of a specific formal model for the system being analyzed, and therefore can be potentially applied to analyzing complex dynamical systems such as generalized semi-Markov processes (GSMPs), for which symbolic and numerical methods are impractical. However, the Younes and Simmons' approach assumes that the system is controllable (not black-box) and can be used to generate sample executions from any state on need basis.

Both the numerical and the current statistical methods suffer from several serious drawbacks when it comes to analyzing practical systems. First, modern day systems are large heterogeneous, and assembled by integrating equipment and software from diverse vendors, making the construction of a formal model of the entire system often impossible and thus limiting the feasibility of numerical and symbolic methods. Second, for large network systems, meaningful experiments may involve dozens or even thousands of routers and hosts, which would mean that the system needs to be deployed before reasonable performance measures can be obtained. However, once they are deployed, such systems cannot be controlled to generate traces from any state, making it impossible to generate execution samples on a need basis as is required by the Younes *et. al*'s statistical approach.

Despite the success of current analysis methods [10, 13, 12, 16, 8], there is therefore a need to develop methods to analyze stochastic processes that can be applied to deployed, unknown "black-box" systems (systems from which traces cannot be generated from any state on need) [1]. In this paper we address these concerns by proposing a new statistical approach to model checking. Like in Younes *et. al*'s approach, discrete event simulation methods are used to obtain a set of sample executions; however, unlike their method we assume no control over the set of samples we obtain. We then test these samples using various statistical tests determined by the property to be verified. Since we assume that the samples are generated before testing, our algorithm relies on statistical hypothesis testing, rather than sequential hypothesis testing. Our inability to generate samples of our choosing and at the time of our choosing ensures that our approach differs from the previous statistical approach in one significant way: unlike the previous approach where the model checker's answer can be guaranteed to be correct within the required error bounds, we instead compute a quantitative measure of confidence (the p-value, in statistical testing terminology [11]) in the model checker's answer. Our algorithm computes the satisfaction of the desired formula by recursively determining the satisfaction of its subformulas (and the confidence of such an answer) in the "states" present in the sample. This presents a technical challenge because our algorithm, being statistical in nature, may be uncertain about the satisfaction of some formulas based on the given samples. The algorithm needs to compute useful answers (as far as possible) even in the presence of uncertain answers about the satisfaction of subformulas. We overcome this challenge by interpreting such "don't know" answers in an adversarial fashion. Our algorithm, thus checks if the sample provides evidence for the satisfaction or violation of a property and the confidence with which such an assertion holds, or gives up and says "don't know." The algorithm that we propose suffers from one drawback when compared with the previous statistical approach. Since we analyze a fixed sample, we will get useful answers only when there are sufficient samples for each "relevant state." Therefore our method is likely to work well only when a finite set of samples is enough to provide sufficient

[1] We assume that the samples generated from the system by discrete event simulation have information about the "system state". We, however, make no assumptions about the transition structure of the underlying system, nor do we assume knowledge about the transition probabilities; the system under investigation is black-box in this sense.

information about relevant states. Examples of systems we can successfully analyze are Continuous-time Markov Chains (CTMCs) or systems whose relevant states are discrete, while we are unlikely to succeed for GSMPs in general.

A closely related approach to analyzing stochastic systems based on Monte Carlo simulation is by Herault et. al. [7], which can model-check discrete-time Markov chains against properties expressed in an expressively weak logic ("positive LTL").

We have implemented the whole procedure in Java as a prototype tool, called VeStA (VErification based on STatistical Analysis).[2] We have experimented with VeStA by applying it to some examples that have been previously analyzed in [16] and the results are encouraging. However, we suspect that VeStA would require a lot more space, because it stores the entire collection of samples it is analyzing. Even though space was not a problem for the examples we tried, we suspect that it may become an issue later.

The rest of the paper is organized as follows. Section 2 defines the class of systems we analyze and the logic we use. In Section 3, we present our algorithm based on statistical hypothesis testing in detail. Details about VeStA and our case studies are presented in Section 4. In Section 5, we conclude and present possible directions for future research.

2 Preliminaries

2.1 Sample Execution Paths

The verification method presented here can be independent of the system model as long as we can generate sample execution paths of the system; the model and its definition are very similar to [17]. We will assume that the system being analyzed is some discrete event system that occupies some state $s \in S$, where S is the set of states of the system. The states in S that can effect the satisfaction of a property of our interest are called the "relevant states." Note that the number of relevant states may be quite small compared to the whole state space of the system. For example, for a formula $\phi_1 \mathcal{U}^{\leq t} \phi_2$ the states that can be reached within time t are relevant. We assume that each relevant state can be uniquely identified and that information about a state's identity is available in the executions. Since samples are generated before running our analysis algorithm, we require that a Monte Carlo simulation is likely to generate a sample that has enough "information" about the relevant states; if not our algorithm is likely to say that it cannot infer anything about the satisfaction of the property.

We assume that there is a labeling function L that assigns to each state a set of atomic propositions (from among those appearing in the property of interest) that hold in that state; thus $L : S \to 2^{AP}$, where AP is a set of relevant atomic propositions. The system remains in a state s until an event occurs, and then proceeds instantaneously to a state s'. An execution path that appears in our sample is thus a sequence

$$\pi = s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} \cdots$$

[2] Available from `http://osl.cs.uiuc.edu/~ksen/vesta/`

where s_0 is the unique initial state of the system, s_i is the state of the system after the ith event and t_i is the time spent in state s_i. If the kth state of this sequence is absorbing, then $s_i = s_k$ and $t_i = \infty$ for all $i \geq k$.

We denote the ith state in an execution π by $\pi[i] = s_i$ and the time spent in the ith state by $\delta(\pi, i)$. The time at which the execution enters state $\pi[i+1]$ is given by $\tau(\pi, i+1) = \sum_{j=0}^{j=i} \delta(\pi, j)$. The state of the execution at time t (if the sum of sojourn times in all states in the path exceeds t), denoted by $\pi(t)$, is the smallest i such that $t \leq \tau(\pi, i+1)$. We let $Path(s)$ be the set of executions starting at state s. We assume $Path(s)$ is a measurable set (in an appropriate σ-field) and has an associated probability measure.

2.2 Continuous Stochastic Logic

Continuous stochastic logic (CSL) is introduced in [1] as a logic to express probabilistic properties of continuous time Markov chains (CTMCs). In this paper we adopt a sublogic of CSL (excluding unbounded untils and stationary state operators) as in [17]. This logic excludes the steady-state probabilistic operators and the unbounded until operators. We next present the syntax and the semantics of the logic.

CSL Syntax

$$\phi ::= true \mid a \in AP \mid \neg\phi \mid \phi \wedge \phi \mid \mathcal{P}_{\bowtie p}(\psi)$$
$$\psi ::= \phi\, \mathcal{U}^{\leq t} \phi \mid \mathbf{X}\phi$$

where AP is the set of atomic propositions, $\bowtie \in \{<, \leq, >, \geq\}$, $p \in [0, 1]$, and $t \in \mathbb{R}_{\geq 0}$. Here ϕ represents a *state* formula and ψ represents a *path* formula. The notion that a state s (or a path π) *satisfies* a formula ϕ is denoted by $s \models \phi$ (or $\pi \models \phi$), and is defined inductively as follows:

CSL Semantics

$$\begin{array}{llll}
s \models true & & s \models a & \text{iff } a \in AP(s) \\
s \models \neg\phi & \text{iff } s \not\models \phi & s \models \phi_1 \wedge \phi_2 & \text{iff } s \models \phi_1 \text{ and } s \models \phi_2 \\
s \models \mathcal{P}_{\bowtie p}(\psi) & \text{iff } Prob\{\pi \in Path(s) \mid \pi \models \psi\} \bowtie p & & \\
\pi \models \mathbf{X}\phi & \text{iff } \tau(\pi, 1) < \infty \text{ and } \pi[1] \models \phi & & \\
\pi \models \phi_1\, \mathcal{U}^{\leq t} \phi_2 & \text{iff } \exists x \in [0, t].\ (\pi(x) \models \phi_2 \text{ and } \forall y & \in [0, x).\ \pi(y) \models \phi_1) &
\end{array}$$

A formula $\mathcal{P}_{\bowtie p}(\psi)$ is satisfied by a state s if $Prob$[path starting at s satisfies ψ] $\bowtie p$. To define probability that a path satisfies ψ we need to define a σ-algebra over the set of paths starting at s and a probability measure on the corresponding measurable space in a way similar to [5]. The path formula $\mathbf{X}\phi$ holds over a path if ϕ holds at the second state on the path. The formula $\phi_1\, \mathcal{U}^{\leq t} \phi_2$ is true over a path π if ϕ_2 holds in some state along π at a time $x \in [0, t]$, and ϕ holds along all prior states along π. This can also be recursively defined as follows:

$$\begin{aligned}
&Sat(s_i \xrightarrow{t_i} \pi_{i+1}, \phi_1\, \mathcal{U}^{\leq t} \phi_2) \\
&\qquad = (t \geq 0) \wedge (Sat(s_i, \phi_2) \vee (Sat(s_i, \phi_1) \wedge Sat(\pi_{i+1}, \phi_1\, \mathcal{U}^{\leq t - t_i} \phi_2)))
\end{aligned} \tag{1}$$

where $Sat(s, \phi)$ (or $Sat(\pi, \psi)$) are the propositions that $s \models \phi$ (or $\pi \models \psi$). This definition will be used later to describe the algorithm for verification of $\phi_1 \mathcal{U}^{\leq t} \phi_2$ formula.

3 Algorithm

In what follows we say that $s \models_{\mathcal{A}} \phi$ if and only if our algorithm (denoted by $\mathcal{A}$) says that ϕ holds at state s. Since the algorithm is statistical, the decision made by the algorithm provides *evidence* about the actual fact. In our approach, we bound the strength of this evidence quantitatively by a number in $[0, 1]$ which gives the probability of making the decision given that the decision is actually incorrect. In statistics, this is called the p-value of the testing method. We denote it by α[3] and write $s \models \phi$ if the state s actually satisfies ϕ.

We assume that we are given a set of finite executions. The length of a finite execution path must be large enough so that all the bounded until formulas can be evaluated on that path. Given a set of sample execution paths starting at the initial state and a formula ϕ, the algorithm works recursively as follows:

verifyAtState$(\phi, s)\{$
  if *cache* contains (ϕ, s) **return** *cache*(ϕ, s);
  else if $\phi = true$ **then** $(z, \alpha) \leftarrow (1, 0.0)$;
  else if $\phi = a \in AP$ **then** $(z, \alpha) \leftarrow$ *verifyAtomic*(a, s);
  else if $\phi = \neg\phi'$ **then** $(z, \alpha) \leftarrow$ *verifyNot*$(\neg\phi', s)$;
  else if $\phi = \phi_1 \wedge \phi_2$ **then** $(z, \alpha) \leftarrow$ *verifyAnd*$(\phi_1 \wedge \phi_2, s)$;
  else if $\phi = \mathcal{P}_{\bowtie p}(\psi)$ **then** $(z, \alpha) \leftarrow$ *verifyProb*$(\mathcal{P}_{\bowtie p}(\psi), s)$;
  store $(s, \phi) \mapsto (z, \alpha)$ in *cache*;
  return (z, α);
$\}$

where *verifyAtState* returns a pair having 0, 1, or *undecided* corresponding to the cases $s \models_{\mathcal{A}} \phi$, $s \not\models_{\mathcal{A}} \phi$, or $\mathcal{A}$ *cannot decide* respectively, as the first component, and p-value for this decision as the second component. To verify a system we check if the given formula holds at the initial state. Once computed, we store the decision of the algorithm for ϕ at state s in a cache to avoid recomputation. The result of our hypothesis testing can be shown to hold in presence of caching. This results in a significantly faster running time and a reduction in the sample set size. In the remainder of this section we define the various procedures *verifyAtomic*, *verifyAnd*, *verifyNot*, and *verifyProb* recursively.

The key idea of the algorithm is to statistically verify the probabilistic operator. We present the corresponding procedure *verifyProb* below.

3.1 Probabilistic Operator

We use statistical hypothesis testing [11] to verify a probabilistic property $\phi = \mathcal{P}_{\bowtie p}(\psi)$ at a given state s. Without loss of generality we show our procedure for $\phi = \mathcal{P}_{\geq p}(\psi)$. This is because, for the purpose of statistical analysis, $\mathcal{P}_{<p}(\psi)$ is essentially the same as $\neg\mathcal{P}_{\geq 1-p}(\psi)$ and $<$ (or $>$) is in effect the same as $\leq$ (or $\geq$). Let p' be the probability that ψ holds over paths starting at s. We say that $s \models \mathcal{P}_{\geq p}(\psi)$ if and only if $p' \geq p$ and $s \not\models \mathcal{P}_{\geq p}(\psi)$ if and only if $p' < p$. We want to decide either $s \models \mathcal{P}_{\geq p}(\psi)$ or $s \not\models \mathcal{P}_{\geq p}(\psi)$. Accordingly we set up two experiments. In the first experiment, we use sample execution paths starting

[3] This should not be confused with the Type I error which is also denoted by α

at s to test the *null hypothesis* $H_0\colon p' < p$ against the *alternative hypothesis* $H_1\colon p' \geq p$. In the second experiment, we test the *null hypothesis* $H_0\colon p' \geq p$ against the *alternative hypothesis* $H_1\colon p' < p$.[4]

Let the number of sample execution paths having a state s somewhere in the path be n. We can treat the portion of all these paths starting at s (suffix) as samples from $Path(s)$. Let $X_1, X_2, \ldots, X_n$ be a random sample having Bernoulli distribution with unknown parameter $p' \in [0,1]$ i.e. for each $i \in [1,n]$, $Prob[X_i = 1] = p'$. Then the sum $Y = X_1 + X_2 + \ldots + X_n$ has binomial distribution with parameters n and p'. We say that x_i, an observation of the random variable X_i, is 1 if the i^{th} sample execution path satisfies ψ and 0 otherwise. In the first experiment, we reject $H_0\colon p' < p$ and say $s \models_{\mathcal{A}} \mathcal{P}_{\geq p}(\psi)$ if $\frac{\sum x_i}{n} \geq p$ and calculate the p-value as $\alpha = Prob[s \models_{\mathcal{A}} \phi \mid s \not\models \phi] = Prob[Y \geq \sum x_i \mid p' < p]$. Note we do not know p'. Therefore, to calculate α we use p which is an upper bound for p'. If we are not able to reject H_0 in the first experiment then we do the second experiment. In the second experiment, we reject $H_0\colon p' \geq p$ and say $s \not\models_{\mathcal{A}} \mathcal{P}_{\geq p}(\psi)$ if $\frac{\sum x_i}{n} < p$ and calculate the p-value as $\alpha = Prob[s \not\models_{\mathcal{A}} \phi \mid s \models \phi] = Prob[Y < \sum x_i \mid p' \geq p]$. Thus, a smaller α represents a greater confidence in the decision of the algorithm $\mathcal{A}$.

3.2 Nested Probabilistic Operators

The above procedure for hypothesis testing works if the truth value of ψ over an execution path determined by the algorithm is the same as the actual truth value. However, in the presence of nested probabilistic operators in ψ, $\mathcal{A}$ cannot determine the satisfaction of ψ over a sample path exactly. Therefore, in this situation we need to modify the hypothesis test so that we can use the inexact truth values of ψ over the sample paths.

Let the random variable X be 1 if a sample execution path π actually satisfies ψ in the system and 0 otherwise. Let the random variable Z be 1 for a sample execution path π if $\pi \models_{\mathcal{A}} \psi$ and 0 otherwise. In our procedure we cannot get samples from the random variable X; instead our samples come from the random variable Z. Let X and Z have Bernoulli distributions with parameters p' and p'' respectively. Let $Z_1, Z_2, \ldots, Z_n$ be a random sample from the Bernoulli distribution with unknown parameter $p'' \in [0,1]$. We say that z_i, an observation of the random variable Z_i, is 1 if *the algorithm says* that the i^{th} sample execution path satisfies ψ and 0 otherwise.

For the formula $\phi = \mathcal{P}_{\geq p}(\psi)$ we calculate regions of indifference, denoted by the fractions δ_1 and δ_2, based on the algorithm's decision for the satisfaction of ψ over the different sample paths. Depending on the values of δ_1 and δ_2 we set up the two experiments. In the first experiment, we test the null hypothesis $H_0\colon p'' \leq p + \delta_1$ against the alternative hypothesis $H_1\colon p'' > p + \delta_1$. If we get $\frac{\sum z_i}{n} > p + \delta_1$ we reject H_0 and say $s \models_{\mathcal{A}} \mathcal{P}_{\geq p}(\psi)$ with p-value $\alpha = Prob[s \models_{\mathcal{A}} \phi \mid s \not\models \phi] = Prob[\sum Z_i > \sum z_i \mid p'' \leq p + \delta_1]$. If we fail to reject H_0 we

[4] While handling nested probabilistic operators, these experiments will no longer be symmetric. Moreover, setting up these two experiments is an alternate way of getting at a conservative estimate of what Type II error (β value) may be.

go for the second experiment, in which the null hypothesis $H_0 \colon p'' \geq p - \delta_2$ against the alternative hypothesis $H_1 \colon p'' < p - \delta_2$. We reject H_0 and say that $s \not\models_{\mathcal{A}} \mathcal{P}_{\geq p}(\psi)$ if $\frac{\sum z_i}{n} < p - \delta_2$ and calculate the p-value as $\alpha = Prob[s \not\models_{\mathcal{A}} \phi \mid s \models \phi] = Prob[\sum Z_i < \sum z_i \mid p'' \geq p - \delta_2]$. Otherwise, we say the algorithm cannot decide.

We now show how to calculate δ_1 and δ_2. Using the samples from Z we can estimate p''. However, we need an estimation for p' in order to decide whether $\phi = \mathcal{P}_{\geq p}(\psi)$ holds in state s or not. To get an estimate for p' we note that the random variables X and Z are related as follows:

$$Prob[Z = 0 \mid X = 1] \leq \alpha' \qquad Prob[Z = 1 \mid X = 0] \leq \alpha'$$

where α' is the p-value calculated while verifying the formula ψ. By elementary probability theory, we have

$$Prob[Z = 1] = Prob[Z = 1 \mid X = 0]Prob[X = 0] + Prob[Z = 1 \mid X = 1]Prob[X = 1]$$

Therefore, we can approximate $p'' = Prob[Z = 1]$ as follows:

$$\begin{aligned} &Prob[Z = 1] \leq \alpha'(1 - p') + 1.p' = p' + (1 - p')\alpha' \\ &Prob[Z = 1] \geq Prob[Z = 1 \mid X = 1]Prob[X = 1] \geq (1 - \alpha')p' = p' - \alpha'p' \end{aligned}$$

This gives the following range in which p'' lies:

$$p' - \alpha'p' \leq p'' \leq p' + (1 - p')\alpha'$$

Hence, $Prob[\sum Z_i > \sum z_i \mid p' \leq p] \leq Prob[\sum Z_i > \sum z_i \mid p' = p] \leq Prob[\sum Z_i > \sum z_i \mid p'' = p - \alpha'p]$ which gives $\delta_2 = \alpha'p$. Similarly, we get $\delta_1 = (1 - p)\alpha'$.

Note that the p-value obtained while verifying ψ over different sample paths are different. We take the worst or the maximum of all such p-values as α'. Moreover, since $\mathcal{A}$ can say *true*, *false*, or *cannot decide*, note that the $\mathcal{A}$ may not have a definite *true* or *false* answer along certain sample paths. For such paths the algorithm will assume the worst possible answer in the two experiments. For the first experiment, where we check whether $\frac{\sum z_i}{n} > p + \delta_1$, we take the answers for the sample paths for which $\mathcal{A}$ cannot decide as *false* and the p-value as 0. For the second experiment we consider the answer for the undecided paths to be *true* and the p-value as 0. This allows us to obtain useful answers even when the sample does not have enough statistical evidence for the satisfaction of a subformula.

Thus we can define the procedure $verifyProb(\mathcal{P}_{\geq p}(\psi), s)$ as follows:

$verifyProb(\mathcal{P}_{\geq p}(\psi), s)\{$
  $zsum_{min} \leftarrow 0$; $zsum_{max} \leftarrow 0$; $\alpha' \leftarrow 0.0$; $n \leftarrow 0$;
  for each sample path π starting at $s\{$
    $(z, \alpha'') \leftarrow verifyPath(\psi, \pi)$;
    if $z =$ *undecided* **then** $\{zsum_{min} \leftarrow zsum_{min} + 1$; $zsum_{max} \leftarrow zsum_{max} + 0;\}$
    else $\{zsum_{min} \leftarrow zsum_{min} + z$; $zsum_{max} \leftarrow zsum_{max} + z;\}$
    $\alpha' \leftarrow max(\alpha', \alpha'')$; $n \leftarrow n + 1$;

```
  }
  if zsum_max/n > p + (1 − p)α′ then
    return (1, Prob[∑ Z_i > zsum_max | p″ = p + (1 − p)α′]);
  else if zsum_min/n < p − pα′ then
    return (0, Prob[∑ Z_i < zsum_min | p″ = p − pα′]);
  else return (undecided, 0.0);
}
```

One can calculate $Prob[\sum Z_i > zsum_{max} \mid p'' = p+(1-p)\alpha']$ (or $Prob[\sum Z_i > zsum_{min} \mid p'' = p - p\alpha'])$ by noting that $\sum Z_i$ has binomial distribution with parameters $p + (1 - p)\alpha'$ (or $p - p\alpha'$) and n.

3.3 Negation

For the verification of a formula $\neg\phi$ at a state s, we recursively verify ϕ at state s. If $s \models_{\mathcal{A}} \phi$ with p-value α we say that $s \not\models_{\mathcal{A}} \neg\phi$ with p-value $Prob[s \not\models_{\mathcal{A}} \neg\phi \mid s \models \neg\phi] = Prob[s \models_{\mathcal{A}} \phi \mid s \not\models \phi] = \alpha$. Similarly, if $s \not\models_{\mathcal{A}} \phi$ with p-value α then $s \models_{\mathcal{A}} \neg\phi$ with p-value α. Otherwise, if $\mathcal{A}$ cannot answer the satisfaction of ϕ at s, we say that $\mathcal{A}$ cannot answer the satisfaction of $\neg\phi$ at s. Thus we can define *verifyNot* as follows:

```
verifyNot(¬φ′, s){
  (z, α) ← verifyAtState(φ′, s);
  if z =undecided then return (undecided, 0.0); else return (1 − z, α);
}
```

3.4 Conjunction

To verify a formula $\phi = \phi_1 \wedge \phi_2$ at a state s, we verify ϕ_1 and ϕ_2 at the state s separately. Depending on the outcome of the verification of ϕ_1 and ϕ_2, $\mathcal{A}$ decides for ϕ as follows:

1. If $s \models_{\mathcal{A}} \phi_1$ and $s \models_{\mathcal{A}} \phi_2$ with p-values α_1 and α_2 respectively, then $s \models_{\mathcal{A}} \phi$. The p-value for this decision is $Prob[s \models_{\mathcal{A}} \phi_1 \wedge \phi_2 \mid s \not\models \phi_1 \wedge \phi_2]$. Note that $s \not\models \phi_1 \wedge \phi_2$ holds in three cases, namely 1) $s \not\models \phi_1$ and $s \models \phi_2$, 2) $s \models \phi_1$ and $s \not\models \phi_2$, and 3) $s \not\models \phi_1$ and $s \not\models \phi_2$. Thus, the p-value for the decision of $s \models_{\mathcal{A}} \phi$ can be taken as the maximum of the p-values in the above three cases which is $max(\alpha_1, \alpha_2)$.
2. If $s \not\models_{\mathcal{A}} \phi_1$ (or $s \not\models_{\mathcal{A}} \phi_2$) and either $s \models_{\mathcal{A}} \phi_2$ or $\mathcal{A}$ cannot decide ϕ_2 at s (or either $s \models_{\mathcal{A}} \phi_1$ or $\mathcal{A}$ cannot decide ϕ_1 at s), then $s \not\models_{\mathcal{A}} \phi$ and the p-value is $Prob[s \not\models_{\mathcal{A}} \phi_1 \wedge \phi_2 \mid s \models \phi_1$ and $s \models \phi_2] = \alpha_1 + \alpha_2$ (or $\alpha_2 + \alpha_1$).
3. If $s \not\models_{\mathcal{A}} \phi_1$ and $s \not\models_{\mathcal{A}} \phi_2$ then $s \not\models_{\mathcal{A}} \phi_1 \wedge \phi_2$. The p-value for this decision is $Prob[s \not\models_{\mathcal{A}} \phi_1 \wedge \phi_2 \mid s \models \phi_1$ and $s \models \phi_2] \leq \alpha_1 + \alpha_2$.
4. Otherwise, $\mathcal{A}$ cannot decide ϕ.

Thus we can define the procedure *verifyAnd* as follows:

```
verifyAnd(φ1 ∧ φ2, s){
  (z1, α1) ← verifyAtState(φ1, s); (z2, α2) ← verifyAtState(φ2, s);
```

```
  if z1 = 1 and z2 = 1 then return (1,max(α1, α2));
  else if z1 = 0 and z2 ≠ 0 then return (0, α1 + α2);
  else if z1 ≠ 0 and z2 = 0 then return (0, α1 + α2);
  else if z1 = 0 and z2 = 0 then return (0, α1 + α2);
  else return (undecided, 0.0);
}
```

3.5 Atomic Proposition

In this simplest case, given $\phi = a$ and a state s, $\mathcal{A}$ checks if $s \models_{\mathcal{A}} a$ or not by checking if $a \in AP(s)$ or not. If $a \in AP(s)$ then $s \models_{\mathcal{A}} \phi$ with p-value 0. Otherwise, $s \not\models_{\mathcal{A}} \phi$ with p-value 0.

```
verifyAtomic(a, s){
  if a ∈ AP(s) then return (1, 0.0); else return (0, 0.0);
}
```

3.6 Next

To verify a path formula $\psi = \mathbf{X}\phi$ over a path π, $\mathcal{A}$ verifies ϕ at the state $\pi[1]$. If $\pi[1] \models_{\mathcal{A}} \phi$ with p-value α then $\pi \models_{\mathcal{A}} \psi$ with p-value α. Otherwise, if $\pi[1] \not\models_{\mathcal{A}} \phi$ with p-value α then $\pi \not\models_{\mathcal{A}} \psi$ with the same p-value. Thus we can define *verifyPath* for $\mathbf{X}\phi$ as follows:

```
verifyPath(Xφ, π){
  return verifyAtState(φ, π[1]);
}
```

3.7 Until

Let $\psi = \phi_1 \,\mathcal{U}^{\leq t}\phi_2$ be an until formula that we want to verify over the path π. We can recursively evaluate the truth value of the formula over the path π by following the recursive definition given by Equation 1. Given the truth value and the p-value for the formula $\phi_1\mathcal{U}^{\leq t'-t_i}\phi_2$ over the suffix π_{i+1}, we can calculate the truth value of $\phi_1\,\mathcal{U}^{\leq t'}\phi_2$ over the path π_i by applying the decision procedure for conjunction and negation. Observe that the recursive formulation in Equation 1 can be unrolled to obtain an equation purely in terms of conjuction and negation (and without any until formulas); it is this "unrolled" version that is used in the implementation for efficiency reasons.

4 Implementation and Performance

We have implemented the above algorithm as part of a prototype Java tool called VeStA. We successfully used the tool to verify several programs having a CTMC model.[5] The performance of the verification procedure depends on the number of samples required to reach a decision with sufficiently small p-value. To get a smaller p-value the number of samples needs to be increased. We need a lot of samples only when the actual probability of a path from a state s satisfying a

[5] We selected systems with CTMC model so that we can compare our results with that of existing tools.

formula ψ is very close to the threshold p in a formula $\mathcal{P}_{\bowtie p}(\psi)$ whose satisfaction we are checking at s.

To evaluate the performance and effectiveness of our implementation we did a few case studies. We mostly took the stochastic systems used for case studies in [16]. The experiments were done on a 1.2 GHz Mobile Pentium III laptop running Windows 2000 with 512 MB memory.[6] We did not take into account the time required for generating samples: we assumed that such samples come from a running system. However, this time, as observed in some of our experiments, is considerably less than the actual time needed for the analysis. We generated samples of length sufficient to evaluate all the time-bounded until formulas. In all of our case studies we checked the satisfaction of a given formula at the initial state. We give a brief description of our case studies below followed by our results and conclusions. The details for the case studies can be obtained from `http://osl.cs.uiuc.edu/~ksen/vesta/`.

Grid World: We choose this case study to illustrate the performance of our tool in the presence of nested probabilistic operators. It consists of an $n \times n$ grid with a robot located at the bottom-left corner and a janitor located at the top-right corner of the grid. The robot first moves along the bottom edge of a cell square and then along the right edge. The time taken by the robot to move from one square to another is exponentially distributed. The janitor also moves randomly over the grid. However, either the robot or the janitor cannot move to a square that is already occupied by the other. The robot also randomly sends a signal to the base station. The underlying model for this example is a CTMC. The aim of the robot is to reach the top-right corner in time T_1 units with probability at least 0.9, while maintaining a minimum 0.5 probability of communication with the base station with periodicity less than T_2 units of time. This can be specified using the CSL formula $\mathcal{P}_{\geq 0.9}(\mathcal{P}_{\geq 0.5}(true\,\mathcal{U}^{\leq T_2}\,communicate)\,\mathcal{U}^{\leq T_1}\,goal)$.

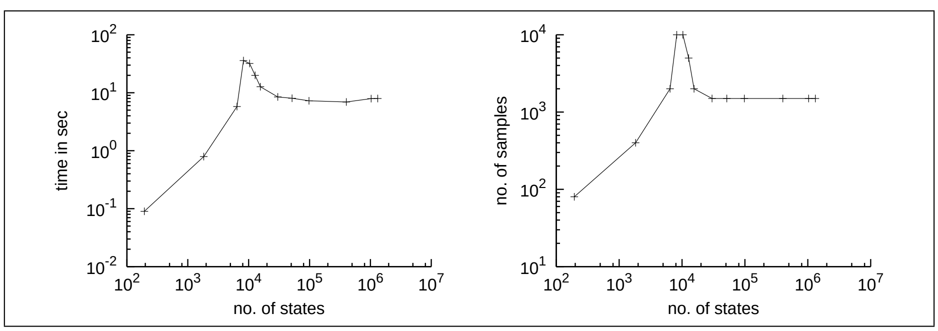

Fig. 1. Grid world: verification time and number of samples versus number of states.

We verified the CSL property for Grid World with $n \in [1, 100]$. The property holds only for $n \in [1, 13]$. The state space of the program is $\Theta(n^3)$. In Fig.1 we

[6] [16] used a 500 MHz Pentium III. However, our performance gain due to the use of faster processor is more than offset by the use of Java instead of C.

plot the results of our experiment. The graph shows that for n closer to 13 the running time and the number of samples required increases considerably to get a respectable p-value of around 10^{-8}. This is because at $n = 13$ the probability that $\mathcal{P}_{\geq 0.5}(\mathit{true}\,\mathcal{U}^{\leq T_2}\,\mathit{communicate})\,\mathcal{U}^{\leq T_1}\,\mathit{goal}$ holds over an execution path becomes very close to 0.9. We found that our graphs are similar to [16].

Cyclic Polling System: This case study is based on a cyclic server polling system, taken from [12]. The model is represented as a CTMC. We use N to denote the number of stations handled by the polling server. Each station has a single-message buffer and they are cyclically attended by the server. The server serves the station i if there is a message in the buffer of i and then moves on to poll the station $(i+1)$ modulo N. Otherwise, the server starts polling the station $i+1$ modulo N. The polling and service times are exponentially distributed. The state space of the system is $\Theta(N.2^N)$. We verified the property that "once a job arrives at the first station, it will be polled within T time units with probability at least 0.5." The property is verified at the state in which all the stations have one message in their message buffer and the server is serving station 1. In CSL the property can be written as $(m_1 = 1) \rightarrow \mathcal{P}_{\geq 0.5}(\mathit{true}\,\mathcal{U}^{\leq T}(s = \mathit{1} \wedge a = \mathit{0}))$, where $m_1 = 1$ means there is one message at station 1, and $s = 1 \wedge a = 0$ means that the server is polling station 1.

Tandem Queuing Network: This case study is based on a simple tandem queuing network studied in [9]. The model is represented as a CTMC which consists of a M/Cox$_2$/1-queue sequentially composed with a M/M/1-queue. We use N to denote the capacity of the queues. The state space is $\Theta(N^2)$. We verified the CSL property $\mathcal{P}_{<0.5}(\mathit{true}\,\mathcal{U}^{\leq T}\mathit{full})$ which states that the probability of the queuing network becoming full within T time units is less than 0.5.

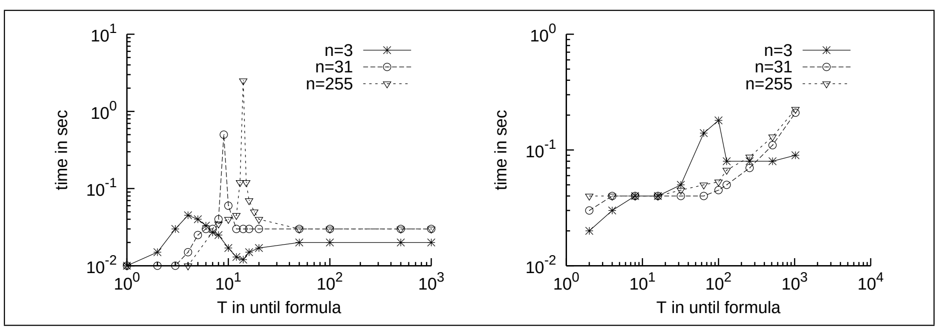

Fig. 2. Polling System and Tandem Queuing Network: Running time versus the parameter T in CSL formula.

The results of the above two case studies is plotted in Fig. 2. The characteristics of the graphs for both the examples are similar to that in [16]. However, while achieving a level of confidence around 10^{-8}, the running time of our tool *for these cases* is faster than the running time of the tool described in [16]. It is important to observe that, unlike the work of [16], VeStA cannot guarantee that the error of it's probabilistic answer is bounded; the p-value computed depends on the specific sample.

We could not compare the number of samples for these case studies as they are not available from [16]; theoretically sequential hypothesis testing should require a smaller sample size than simple hypothesis testing to achieve the same level of confidence. While in our case studies we never faced a memory problem, we suspect that this may be a problem in very big case studies. We observed that, although the state space of a system may be large, the number of states that appeared in the samples may be considerably smaller. This gives us hope that our approach may work as well for very large-scale systems.

5 Conclusion and Future Work

We have presented a new statistical approach to verifying stochastic systems based on Monte Carlo simulation and statistical hypothesis testing. The main difference between our approach and previous statistical approaches is that we assume that the system under investigation is not under our control. This means that our algorithm computes on a fixed set of executions and cannot obtain samples as needed. As a consequence, the algorithm needs to check for the satisfaction of a property and compute the p-value of its tests. Since the sample may not provide sufficient statistical evidence to conclude the satisfaction or violation of a property, one technical challenge is to provide useful answers despite insufficient statistical evidence for the satisfaction of subformulas. We implemented our approach in Java and our experimental case studies have demonstrated that the running time of our tool is faster than previous methods for analyzing stochastic systems in at least some cases; this suggests that our method may be a feasible alternative.

Another important challenge is the amount of memory needed. Since we store all the sample executions, our method is memory intensive, and though we did not suffer from memory problems on the examples studied here, we suspect that it will be an issue when analyzing larger case studies. Hence, there is a need to design efficient data structures and methods to store and compute with a large set of sample executions. We suspect statistical hypothesis testing approaches (as opposed to sequential hypothesis testing approaches) might be extendible to check liveness for certain types of systems, possibly by extracting some additional information about the traces. Note that liveness properties are particularly complicated by the fact that the operators may be nested. We are currently exploring that direction. Finally, it would be interesting to apply statistical methods to analyze properties described in probabilistic logics other than CSL.

Acknowledgments

The work is supported in part by the DARPA IPTO TASK Award F30602-00-2-0586, the DARPA IXO NEST Award F33615-01-C-1907, the DARPA/AFOSR MURI Award F49620-02-1-0325, the ONR Grant N00014-02-1-0715, and the Motorola Grant MOTOROLA RPS #23 ANT. We would also like to acknowledge the contribution of Jose Meseguer to this research. Our work has benefitted considerably from stimulating discussions with him and from our many years of collaboration on probabilistic rewriting theories. We would like to thank Reza Ziaie for reviewing a previous version of this paper and giving us valuable feedback.

References

1. A. Aziz, K. Sanwal, V. Singhal, and R. K. Brayton. Verifying continuous-time Markov chains. In *8th International Conference on Computer Aided Verification (CAV'96)*, volume 1102, pages 269–276. Springer, 1996.
2. R. Alur, C. Courcoubetis, and D. Dill. Model-checking for probabilistic real-time systems (extended abstract). In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming (ICALP'91)*, volume 510 of *LNCS*, pages 115–126. Springer, 1991.
3. A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model-checking continuous-time Markov chains. *ACM Transactions on Computational Logic*, 1(1):162–170, 2000.
4. C. Baier, E. M. Clarke, V. Hartonas-Garmhausen, M. Z. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming (ICALP'97)*, volume 1256 of *LNCS*, pages 430–440. Springer, 1997.
5. C. Baier, J. P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time markov chains. In *International Conference on Concurrency Theory*, volume 1664 of *LNCS*, pages 146–161. Springer, August 1999.
6. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proceedings of 15th Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS'95)*, volume 1026 of *LNCS*.
7. T. Herault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate Probabilistic Model Checking. In *Proceedings of Fifth International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'04)*, volume 2937 of *LNCS*, pages 73–84. Springer, 2004.
8. H. Hermanns, J. P. Katoen, J. Meyer-Kayser, and M. Siegle. A Markov chain model checker. In *Tools and Algorithms for Construction and Analysis of Systems (TACAS'00)*, pages 347–362, 2000.
9. H. Hermanns, J. Meyer-Kayser, and M. Siegle. Multi-terminal binary decision diagrams to represent and analyse continuous-time markov chains. In *Proceedings of 3rd International Workshop on the Numerical Solution of Markov Chains (NSMC'99)*, 1999.
10. J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.
11. R. V. Hogg and A. T. Craig. *Introduction to Mathematical Statistics*. Macmillan, New York, NY, USA, fourth edition, 1978.
12. O. C. Ibe and K. S. Trivedi. Stochastic petri net models of polling systems. *IEEE Journal on Selected Areas in Communications*, 8(9):1649–1657, 1990.
13. M. Z. Kwiatkowska, G. Norman, and D. Parker. Prism: Probabilistic symbolic model checker, 2002.
14. M. Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Verifying quantitative properties of continuous probabilistic timed automata. In *International Conference on Concurrency Theory (CONCUR'00)*, volume 1877 of *LNCS*, pages 123–137. Springer, 2000.
15. K. Sen, M. Viswanathan, and G. Agha. On statistical model checking of probabilistic systems. In *17th conference on Computer Aided Verification (CAV'05)*, LNCS (To Appear). Springer, July 2005.
16. H. L. S. Younes, M. Kwiatkowska, G. Norman, and D. Parker. Numerical vs. statistical probabilistic model checking: An empirical study. In *10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04)*. Springer, 2004.

17. H. L. S. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *14th International Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 223–235. Springer, 2002.

A Soundness of the Algorithm in Section 3

Let us consider a coin toss experiment with $P(H) = p_0$ and $P(T) = 1 - p_0$, where p_0 is unknown. We want to test whether $p_0 \geq p$ or $p_0 < p$. There are two ways to set up the experiment as follows:

Type 1 Assume that $p_0 \notin [p - \delta_1, p + \delta_2]$. Let $H_0\colon p_0 < p$ and $H_a\colon p_0 \geq p$. We accept H_a if $\frac{\sum_{i \in [1,n]} z_i}{n} \geq p$ [7] and accept H_0 otherwise, where n is the sample size and z_is are our observations. Then the sample size for the experiments must be set such that both $Prob[\text{we accept } H_a \mid H_0 \text{ is correct}] \leq \alpha$ and $Prob[\text{we accept } H_0 \mid H_a \text{ is correct}] \leq \beta$ hold. This is how we set up the experiments in [15], and to our understanding this is how experiment is set up in [17]. Note that this experiment is not correct if $p_0 \notin [p - \delta_1, p + \delta_2]$. This is a shortcoming of this approach. We have to choose very small δ_1 and δ_2 to make sure that $p_0 \notin [p - \delta_1, p + \delta_2]$.

Type 2 Let p_0 be any value between 0 and 1; in particular, we do not assume that $p_0 \notin [p - \delta_1, p + \delta_2]$. Let $H_0\colon p_0 < p$ and $H_a\colon p_0 \geq p$. We accept H_a if $\frac{\sum_{i \in [1,n]} z_i}{n} \geq p + \delta_2$ and accept H_0 if $\frac{\sum_{i \in [1,n]} z_i}{n} \leq p - \delta_1$, where n is the sample size and z_is are our observations. We say "don't know" if $p - \delta_1 < \frac{\sum_{i \in [1,n]} z_i}{n} < p + \delta_2$. Then the sample size for the experiments must be set such that both $Prob[\text{we accept } H_a \mid H_0 \text{ is correct}] \leq \alpha$ and $Prob[\text{we accept } H_0 \mid H_a \text{ is correct}] \leq \beta$ hold. This removes the shortcoming of the previous approach that we have to assume that $p_0 \notin [p - \delta_1, p + \delta_2]$. However, in this case we can have "don't know" answers. Note that in this case we can choose any δ_1 and δ_2 without the fear that our experiment may go wrong. However, bigger δ_1 and δ_2 will result in more "don't know" answers.

Henceforth, we will only consider Type 2 experiments. A Type 2 experiment for the coin toss is specified by the 4-tuple $(\alpha, \beta, \delta_1, \delta_2)$; given these parameters we know how many samples we need to draw in a Type 2 experiment, and we know what decision to take once we are given a sample of sufficiently large size. Let us denote by $\text{sample}(\alpha, \beta, \delta_1, \delta_2)$ to be the (minimum) sample size required ensure that the type I and type II errors are bounded by α and β, when conducting an experiment as above. Let E be the set of all experiments of Type 2 such that $1 \geq \alpha = \beta \geq 0$, $p > \delta_1 > 0$, and $(1 - p) > \delta_2 > 0$.

Consider a sample $\hat{z} = z_1, z_2, \dots z_n$ of n coin tosses. Define $E^{\text{yes}}(\hat{z}) \subseteq E$ to be the set of experiments (with type I and type II error bounds being equal) such

[7] The threshold can of course be modified to account for the fact that the median in the binomial distribution is not the same as the mean.

that the sample $\hat{z}$ will cause such a Type 2 experiment to accept the alternate hypothesis. More formally, $e = (\alpha, \beta, \delta_1, \delta_2) \in E^{\text{yes}}(\hat{z})$, when the following holds:

1. $\text{sample}(\alpha, \beta, \delta_1, \delta_2) \leq n$, i.e, $\hat{z}$ is sufficiently large to ensure the type I and type II errors to be bounded by α and β
2. $\frac{\sum_{i=1}^{n} z_i}{n} \geq p + \delta_2$.

Similarly, we can define $E^{\text{no}}(\hat{z}) \subseteq E$ to be the set of experiments that will result in the null hypothesis being accepted. In other words, $e = (\alpha, \beta, \delta_1, \delta_2) \in E^{\text{no}}(\hat{z})$, when

1. $\text{sample}(\alpha, \beta, \delta_1, \delta_2) \leq n$
2. $\frac{\sum_{i=1}^{n} z_i}{n} < p - \delta_1$.

Consider the following algorithm A. Given a sample $\hat{z}$ (drawn by n coin tosses), the algorithm does one of two things: Finds an experiment $(\alpha, \beta, \delta_1, \delta_2) \in E^{\text{yes}}(\hat{z})$ and outputs "yes" and number α; or finds an experiment $(\alpha, \beta, \delta_1, \delta_2) \in E^{\text{no}}(\hat{z})$ and outputs "no" and number β. Let us assume A is a deterministic algorithm (once it is given the sample $\hat{z}$). Observe that

$$\begin{array}{c} Prob[A(\hat{z}) = \text{yes} \mid p_0 < p] \leq \alpha \\ Prob[A(\hat{z}) = \text{no} \mid p_0 \geq p] \leq \beta \end{array}$$

where the probability is being taken over the fact that $\hat{z}$ is a random sample of size n.

Our algorithm, presented in Section 3, can be viewed as doing the same thing as what algorithm A was doing in the context of coin tosses. It is trying to find an "experiment" that will allow it to consistently label the samples. In the case of model checking the "experiment" and sample have more structure. The sample is now a tree. A consistent "experiment" is a labelling of each node in the tree (along with the satisfaction or non-satisfaction of subformulas) with bounds, α, β, δ_1 and δ_2 that will correspond to right labelling, i.e., the number of samples needed to guarantee a particular error bound is less than that in the given sample from that particular state, and the "yes" and "no" labelling is consistent with what an actual run of an algorithm like [17] might require. And just like in the case of the coin toss, the label α of the root (with respect to the property being tested for) gives a bound on the p-value of such an algorithm. Once again, the algorithm is deterministic: given a sample (or tree) the algorithm will label the internal nodes with α etc. in a determined manner. The probability is taken over the fact that the sample is drawn randomly from the given system.