2011

Unbounded-2-bounded: a two-phase approximation for model checking unbounded until properties of probabilistic systems

Paul Jennings
Iowa State University

Unbounded-2-bounded: a two-phase approximation for model checking unbounded until properties of probabilistic systems

by

Paul Jennings

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Samik Basu, Co-major Professor
Arka P. Ghosh, Co-major Professor
Andrew Miner
Ting Zhang

Iowa State University

Ames, Iowa

2011

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 Introduction

A large variety of systems (e.g. communication protocols over lossy channels, client-server protocols with unreliable servers, and distributed leader-election algorithms) exhibit probabilistic behavior in which the systems evolve from one configuration to another following a certain pre-specified probability distribution. Such probabilistic behaviors are often modeled using discrete time Markov chains (DTMC), continuous time Markov chains (CTMC), and Markov decision processes (MDP). Several techniques and tools have been developed to prove the correctness (in a probabilistic sense) of these system models. One such technique that automatically verifies the conformance of probabilistic systems modeled as DTMC, CTMC, or MDP (Roy and Gopinath (2005); Norman and Shmatikov (2006); Duflot et al. (2006); Kwiatkowska et al. (2008)) against desired properties expressed in probabilistic temporal logic (e.g., PCTL from Hansson and Jonsson (1994) or CSL from Aziz et al. (2000)), is called probabilistic model checking.

Broadly, there are two categories of probabilistic model checking methods. The first category, typically referred to as the *numerical method* (see Hansson and Jonsson (1994); Bianco and de Alfaro (1995); Courcoubetis and Yannakakis (1995); Aziz et al. (2000); Baier et al. (2003)) relies on exploration of the entire state-space of the probabilistic system and applies linear equation solvers to obtain the probability with which the system satisfies a property. In contrast, the other method, referred to as the *approximate* or *statistical method* (see Younes and Simmons (2002); Herault et al. (2004); Sen et al. (2005)), samples a finite set of paths in the system and infers the approximate probability of satisfaction of a property by the whole system using statistical arguments and probabilistic analysis. While the numerical method provides exact solutions, it requires complete knowledge of the system and may fail for systems with a large state space (known as the state-space explosion problem). It is in this situation that sampling-based statistical methods are useful. However, two important issues need to be addressed before any statistical verification approach can be applied effectively: the number of sample paths to simulate and the length for each sample path.

The sampling method explores only a portion of the state space of the system and therefore the accuracy of the verification results depends on the size of the sample (i.e., the number of sample paths, N). This value of N is typically obtained from well-known probabilistic bounds that ascertain the closeness of the estimate to the actual probability with respect to certain pre-specified error limit (ϵ) and confidence parameter (δ).

By definition, the sampling method can consider only paths of finite length. This is not a problem when the property under consideration has a specific bound (as for a *bounded* path property): $\varphi_1 \ \mathtt{U}^{\leq k} \ \varphi_2$, i.e., φ_2 must be satisfied within k steps from the start state and in all states before that φ_1 must be satisfied. This implies that finite paths of length k are sufficient to verify such properties.

However, the property of interest may be *unbounded*[1], i.e., $\varphi_1 \ \mathtt{U} \ \varphi_2$. The semantics of the property states that a path satisfies it if and only if there exists a state in the path which satisfies φ_2 and in all states before that state φ_1 is satisfied. Note that φ_1 can be satisfied any number of times in a path before φ_2 is satisfied for the first time. Therefore, in any path of finite length where every state satisfies $\varphi_1 \wedge \neg\varphi_2$, it is impossible to infer whether any extension of the path will eventually satisfy or not satisfy $\varphi_1 \ \mathtt{U} \ \varphi_2$. The existing statistical methods for probabilistic model checking either assume that an appropriate bound is given, as in Herault et al. (2004); Younes and Simmons (2006); or require some specific knowledge of the system behavior, as in Sen et al. (2005); Rabih and Pekergin (2009).

In contrast, we introduce a new statistical method which automatically computes a bound k_0 on simulation path length and does not require any prior knowledge of the system. The central theme of our technique is to reduce the problem of verifying ($\varphi_1 \ \mathtt{U} \ \varphi_2$) to that of its

[1]Grunske Grunske (2008) proposes patterns for specifying probabilistic properties where he discusses the need for (time-) unbounded until properties for representing various probabilistic properties, e.g., invariance, existence, response.

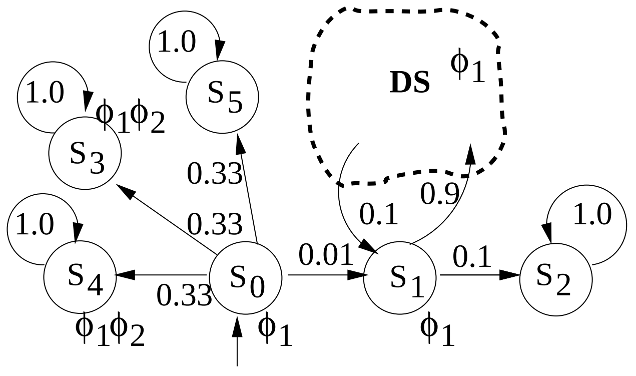


Figure 1.1 A simple example.

bounded counter-part ($\varphi_1 \; \mathtt{U}^{\leq k_0} \; \varphi_2$). The reduction is possible only when a suitable k_0 can be obtained for which $\mathsf{P}(s, \varphi_1 \; \mathtt{U}^{\leq k_0} \; \varphi_2)$ (i.e., the probability of satisfying of $\varphi_1 \; \mathtt{U}^{\leq k_0} \; \varphi_2$ at state s) is a *good* approximation of $\mathsf{P}(s, \varphi_1 \; \mathtt{U} \; \varphi_2)$. In other words, the bound k_0 is large enough to make the difference between $\mathsf{P}(s, \varphi_1 \; \mathtt{U}^{\leq k_0} \; \varphi_2)$ and $\mathsf{P}(s, \varphi_1 \; \mathtt{U} \; \varphi_2)$ small. Such a k_0 provides an approximate upper bound of the sample path length needed for our statistical verification technique. We obtain k_0 using the probability of satisfying a different *bounded* path property: $\psi_k := (\varphi_1 \; \mathtt{U}^{\leq k} \; \varphi_2) \vee (\neg\varphi_2 \; \mathtt{U}^{\leq k} \; (\neg\varphi_1 \wedge \neg\varphi_2))$. This property states that the original property ($\varphi_1 \; \mathtt{U} \; \varphi_2$) is either satisfied (first disjunct) or unsatisfied (second disjunct) in at most k steps. We prove that a suitable k_0 is one for which $\mathsf{P}(s, \psi_{k_0})$ is close to 1, and that the degree of "closeness" is related to ϵ, the overall measure of accuracy of the entire statistical method.

In essence, there are two phases in our method. The first phase estimates $\mathsf{P}(s, \psi_k)$ for $k = 0, 1, 2, \ldots$ and chooses k_0 which satisfies $\mathsf{P}(s, \psi_{k_0}) \geq 1 - \epsilon_0$, where $\epsilon_0 < \epsilon$. In the second phase, $\mathsf{P}(s, (\varphi_1 \; \mathtt{U}^{\leq k_0} \; \varphi_2))$ is estimated, which in turn serves as an estimate of $\mathsf{P}(s, (\varphi_1 \; \mathtt{U} \; \varphi_2))$. The computations for each phase involve only *bounded-path* properties and can be carried out efficiently using the existing sampling techniques such as those described in Herault et al. (2004). For systems where $\mathsf{P}(s, \psi_k) \ngeq 1 - \epsilon_0$ for any k, we propose an alternate heuristic method to estimate $\mathsf{P}(s, (\varphi_1 \; \mathtt{U} \; \varphi_2))$ based on changes in the estimates of $\mathsf{P}(s, \psi_k)$ with k.

1.1 Illustrative Example

To provide an intuitive explanation of why the proposed technique is useful and effective, we present a simple toy example (Figure 1.1) where the proposed method is applied successfully and where both the numerical method and the existing statistical verification method as implemented in the popular PRISM model checker (described in Hinton et al. (2006)) fail. The example contains a probabilistic transition system containing $6 + n$ states where n is some large integer. The state s_0 is the start state of the system. The dotted segment in the figure represents some "complicated" transition structure on n different states (see Jennings et al. (2010) for the specification). We will refer to this segment as DS. Let proposition φ_1 hold in all states except s_2 and s_5, and proposition φ_2 hold in states s_3 and s_4. The objective is to find the probability of satisfying the property $(\varphi_1 \texttt{ U } \varphi_2)$ at state s_0. From the probabilities specified in the figure, we know that the resultant probability is 0.66 as there are only two paths $(s_0, s_3, \ldots)$ and $(s_0, s_4, \ldots)$ that satisfy the property.

We experimented with the PRISM model checker Hinton et al. (2006) using the above example. PRISM's numerical method fails as the large state-space (large n) results in state-space explosion. PRISM's statistical method takes a parameter ϵ as input and provides an approximate result within an ϵ error margin. Our experiments with several $\epsilon > 0$ failed to provide any estimate. This is because PRISM's statistical method requires that the satisfaction of a property $\varphi_1 \texttt{ U } \varphi_2$ be known within some pre-specified number of steps. The failure happens when at least one sample path enters DS and does not leave DS. In this case, φ_1 is satisfied for all states in the path, but φ_2 is not satisfied for any states in the path, i.e., it cannot be known whether $\varphi_1 \texttt{ U } \varphi_2$ is satisfied or not.

In terms of ψ_k as introduced earlier, the above requirement in PRISM's statistical method is equivalent to $\mathsf{P}(s_0, \psi_k) = 1$, for some pre-specified bound k ($k = 10{,}000$ by default in PRISM). In general, it is not possible to find an appropriate value for k such that $\mathsf{P}(s_0, \psi_k) = 1$. In contrast, we claim that it is not necessary to verify whether $\mathsf{P}(s_0, \psi_k)$ is equal to 1. The necessary precision for an approximate statistical model checking can be obtained by identifying a k (k_0 in our terminology) for which $\mathsf{P}(s_0, \psi_k)$ is close to 1. In the above example, such a bound can be immediately obtained as the sample paths $(s_0, s_1, \ldots)$ have very low probability (≤ 0.01). Once such a bound is obtained, we compute $\mathsf{P}(s_0, \varphi_1 \; \mathtt{U}^{\leq k_0} \; \varphi_2)$ which approximately coincides with $\mathsf{P}(s_0, \varphi_1 \; \mathtt{U} \; \varphi_2)$. In our experiments, with $n \approx 10^8$, the PRISM model checker fails to provide any result, while our method identifies a bound $k_0 = 3343$ and estimates the probability to be equal to 0.6601 in approximately 97 seconds.

1.2 Contributions

The contributions of our approach are summarized as follows:

1. *Automation.* We present a methodology for automatically selecting a suitable bound k_0 which allows unbounded until properties for probabilistic systems modeled as Discrete Time or Continuous Time Markov Chains to be verified using the corresponding k_0-bounded until properties. The reduction allows us to re-use the existing results for statistical verification of bounded until properties to identify the bound on the sample size N required to infer results within a pre-specified error margin.

2. *Universal Application.* The technique is applied for probabilistic model checking of any unbounded (untimed) path properties (expressed in PCTL or CSL) for models expressed as DTMC and CTMC.

3. *Theoretical Correctness.* We prove the soundness of our technique and discuss the condition under which our technique will always terminate with an estimate with a pre-specified error bound and confidence parameter. When the required condition is not satisfied in a system, our technique (as well as any other statistical method for probabilistic model checking that does not require prior knowledge of the complete model structure) will fail to terminate. We discuss a heuristic for dealing with such systems.

4. *Effective Implementation and Usability.* We present PRISM-U2B, an optimized implementation of our method based on the well-developed probabilistic model checking tool PRISM. We leverage PRISM's realization of generating sample simulations from a given

DTMC or CTMC model and re-use PRISM's widely-used graphical user interface, command-line interface and input specification languages, thereby reducing the cognitive burden of understanding and using PRISM-U2B. It is worth mentioning that Jansen et al. (2007) rates PRISM as the most user-friendly tool for probabilistic model checking in terms of modeling features and usability. Being based on PRISM, our tool PRISM-U2B enjoys similar ease of use.

5. *Experimental Evaluation.* We compare PRISM-U2B and PRISM's statistical method, and empirically show that PRISM-U2B is about 1.5 times faster than PRISM for the examples where both can compute an estimate. We discuss several examples (and present results) where PRISM fails to provide an estimate while PRISM-U2B successfully terminates with an estimate. We also compare the tools PRISM-U2B and MRMC. MRMC is faster than PRISM-U2B as MRMC, unlike PRISM-U2B, utilizes pre-analysis of the model. However, MRMC fails for case studies with large state-space, where PRISM-U2B successfully computes the result. The tool PRISM-U2B, as well as its documentation and case studies[2] can be obtained at Jennings et al. (2010).

1.3 Organization

Chapter 2 provides a brief overview of discrete time Markov chains and the probabilistic temporal logic PCTL. Chapter 3 discusses related work. Chapter 4 presents our solution methodology and its proof of correctness. The implementation is discussed in Chapter 5. Chapter 6 discusses the necessary condition for the termination of our technique and presents a heuristic method when this condition is not satisfied. Chapter 7 discusses the application of our technique for continuous time Markov chains. Chapter 8 presents a brief summary of the tool PRISM-U2B followed by its empirical evaluation using several examples in Chapter 9. Finally, Chapter 10 concludes with the summary and future avenues of research.

[2]Most of the case studies are available at `http://www.prismmodelchecker.org/casestudies/index.php`.

CHAPTER 2 Preliminaries

We proceed with a brief summary on probabilistic systems modeled as DTMC followed by the syntax and semantics of probabilistic properties expressed in the logic of PCTL. The proposed method and its explanations and theoretical results will be presented in terms of these concepts. Subsequently, we will show (Chapter 7) that the proposed method is equally applicable for specific types of reachability properties in CTMC.

2.1 Probabilistic Systems: Discrete Time Markov Chain Models

We will describe the behavior of a system that evolves from one configuration to another based on a certain probability as a state machine augmented with probabilities labeling the transitions. In its simplest form, where every transition in the state machine represents probabilistic choice and every choice only depends on the current configuration, the representation aligns with the definition of a DTMC.

Definition 1. *A Discrete Time Markov Chain is defined as DTMC* $= (S, s_I, T, L)$*, where:*

- S *is a finite set of states*
- $s_I \in S$ *is the initial or start state*
- $T : S \times S \rightarrow [0, 1]$ *is a transition probability function such that* $\forall s : \sum_{s' \in S} T(s, s') = 1$
- $L : S \rightarrow \mathcal{P}(AP)$ *is the labeling function which labels each state with a set of atomic propositions* $\subseteq AP$ *that hold in that state.*

Paths and Probability Measures. A path in a DTMC, denoted by π, is a finite or infinite sequence of states $(s_0, s_1, s_2, s_3, \ldots)$ such that for all $i \geq 0 :\ s_i \in S$ and $T(s_i, s_{i+1}) > 0$. We denote the set of all infinite paths starting from s as $Path(s)$. $\pi[i]$ denotes the i-th state in the path π and $|\pi|$ is the length of π in terms of the number of transitions in π. For example, for an infinite path π, $|\pi| = \infty$, while for a finite path $\pi = (s_0, \ldots, s_n)$, $|\pi| = n, n \geq 0$. The cylinder set, denoted by $C_s(\pi)$ for a state s and a finite length path π starting from s, is defined as $C_s(\pi) = \{\pi' : \pi' \in Path(s) \ \wedge \ \pi \text{ is prefix of } \pi'\}$. Essentially, $C_s(\pi)$ is the set of all infinite paths $\in Path(s)$ with the common finite length prefix π. For any finite path π with $|\pi| = n$ we define

$$P(\pi) = \begin{cases} 1 \text{ if } n = 0 \\ T(\pi[0], \pi[1]) \times \ldots \times T(\pi[n-1], \pi[n]) \text{ otherwise} \end{cases} \tag{2.1}$$

For a cylinder $C_s(\pi)$, define $Pr(C_s(\pi)) = P(\pi)$. It is well-known that this probability measure $Pr(\cdot)$ extends uniquely over all sets in the relevant $\sigma-$algebra of path(s).

2.2 Probabilistic Properties

Properties of a DTMC can be expressed using PCTL, an extension of standard CTL augmented with probabilistic specifications. Let φ represent a state formula and ψ represent a path formula. Then PCTL syntax is defined as follows:

$$\varphi \quad \rightarrow \quad tt \mid a \in AP \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathtt{P}_{\bowtie \mathtt{r}}(\psi) \quad \text{and} \quad \psi \ \rightarrow \ \varphi \ \mathtt{U} \ \varphi \mid \varphi \ \mathtt{U}^{\leq k} \ \varphi$$

In the above, $\bowtie \in \{\leq, \geq, <, >\}$, $r \in [0, 1]$ and $k \in \{0, 1, \ldots\}$. Note that we always use state formulas to specify the properties of a DTMC and path formulas only occur inside $\mathtt{P}_{\bowtie \mathtt{r}}(.)$. A state s (or a path π) satisfying a state formula φ (or a path formula ψ) is denoted by $s \models \varphi$ (or $\pi \models \psi$), and is inductively defined as follows:

$$\begin{array}{lll} s \models tt \ \text{ for all } s \in S & s \models a \ \Leftrightarrow a \in L(s) & s \models \neg\varphi \ \Leftrightarrow s \not\models \varphi \\ s \models \varphi_1 \wedge \varphi_2 \ \Leftrightarrow s \models \varphi_1 \text{ and } s \models \varphi_2 & & s \models \mathtt{P}_{\bowtie \mathtt{r}}(\psi) \ \Leftrightarrow \mathsf{P}(s, \psi) \bowtie r \end{array}$$

In the above, $\mathsf{P}(s, \psi) = Pr(\{\pi \in Path(s) : \pi \models \psi\})$. In other words, $s \models \mathtt{P}_{\bowtie r}(\psi)$ holds if and only if the probability *that ψ is true for an outgoing infinite path from state s* is $\bowtie r$. For any infinite path π:

$$\pi \models \varphi_1 \ \mathtt{U}^{\leq k} \ \varphi_2 \Leftrightarrow \exists 0 \leq i \leq k : \pi[i] \models \varphi_2 \ \wedge \ \forall j < i : \pi[j] \models \varphi_1$$

$$\pi \models \varphi_1 \ \mathtt{U} \ \varphi_2 \Leftrightarrow \exists i \geq 0 : \pi[i] \models \varphi_2 \ \wedge \ \forall j < i : \pi[j] \models \varphi_1$$

Note that $\varphi_1 \ \mathtt{U} \ \varphi_2 \equiv \exists k : \varphi_1 \ \mathtt{U}^{\leq k} \ \varphi_2$. We refer to properties of the form $\varphi_1 \ \mathtt{U} \ \varphi_2$ as unbounded because the bound k is not fixed and not known a priori.

CHAPTER 3 Related Work

Legay and Delahaye (2010) survey and compare different probabilistic model checking techniques based on statistical sampling. In this section, we elaborate on some of these existing techniques and distinguish the contributions of our technique and those of the existing ones. We proceed with an overview of sampling based methods for probabilistic model checking.

Note that for all of these methods, it is necessary to decide whether a simulation path satisfies or does not satisfy the given property ψ. For a bounded path property $\psi = \varphi_1 \ \mathtt{U}^{\leq k} \ \varphi_2$, simulation paths of length at most k definitely satisfy or do not satisfy the property. For an unbounded path property $\psi = \varphi_1 \ \mathtt{U} \ \varphi_2$, it is not possible to know *a priori* the path length that is necessary to decide whether or not a given path satisfies ψ. In the following discussion, it is assumed that samples can be taken from the system. This implies that only bounded properties can be considered, or that some method exists that can determine the appropriate bound. The different methods to determine the bound (or to determine when to stop sampling) are discussed with each theoretical result.

3.1 Sampling Based Methods for Probabilistic Model Checking

The verification problem of whether a `DTMC` satisfies a probabilistic temporal property expressed in `PCTL` (Chapter 2) can be reduced to the problem of solving a set of linear equations over a set of variables where each variable corresponds to the probability that a state in the `DTMC` satisfies the given temporal (path) property. As mentioned before, this method of verification by solving linear equations numerically is referred to as the numerical method for probabilistic model checking (Hansson and Jonsson (1994); Bianco and de Alfaro (1995); Courcoubetis and Yannakakis (1995); Aziz et al. (2000); Baier et al. (2003)). However, as in traditional model checking, the numerical method suffers from state-space explosion because it requires complete knowledge of the model state space. Furthermore, it is not possible to apply numerical methods in situations where the model transition structure is not available, but rather only simulation runs of the model can be obtained as needed (e.g. black-box systems).

To address the problem of state-space explosion and avoid the necessity of prior complete knowledge of the model transition structure, sampling based methods have been proposed and developed. The central theme of these methods is to infer from sample simulations whether the probabilistic property is satisfied by the model, while controlling the error in the inference using statistical bounds. In this context, there are two related inferencing methods: in one, the inference involves *estimating* the probability with which a state satisfies a property, while in the other, the inference involves *verifying* (accepting, or more precisely, rejecting) whether the probability that a state satisfies a property is greater (or less) than a pre-specified value. The former computes the estimate within certain confidence bounds (defined by error limits), while the latter employs hypothesis testing with an indifference width (interval where the null hypothesis and its alternate remain undecided). Both methods aim to obtain results such that the error in the result can be controlled. The methods use sample simulations associated to random variables $X_1, X_2, \dots$ with outcomes of 1 or 0 depending on whether the simulation satisfies the path property or not; X_i corresponds to the outcome of the i-th simulation. The proportion

$$\hat{p} = p_N = \frac{\sum_{i=1}^{N} X_i}{N} \tag{3.1}$$

is used in both the statistical estimation and the hypothesis testing methods as an estimate of the probability that a property is satisfied.

Since it takes some time to obtain each sample, a secondary goal in any statistical method is to minimize N, which is the number of samples required. Intuitively, taking more samples increases confidence in the result of the test; however, different procedures can achieve the same confidence levels for varying values of N, as shown below.

3.1.1 Hypothesis Testing Based Statistical Estimation

Hypothesis based statistical estimation tests whether the probability that s satisfies ψ is $\bowtie r$, where the probabilistic property under consideration is $\mathtt{P}_{\bowtie r}(\psi)$. For the discussion here, assume that $\bowtie$ is $\geq$. In this setting, the null hypothesis $H_0 : p \geq r$ and its alternate $H_1 : p < r$ are considered. For some number N of sample simulations starting from s, the proportion of paths $\hat{p}$ (Equation 3.1) that satisfy ψ, is computed. The test rejects the null hypothesis if the proportion is less than r, and does not reject the null hypothesis if the proportion is greater than or equal to r. There are two types of errors considered in such a testing procedure. Type I error (false negative error) is the probability that the null hypothesis is rejected when in reality it holds. Type II error (false positive error) is the probability that the null hypothesis is not rejected when in reality it does not hold. The maximal bounds on these probabilities are typically denoted by α and β, i.e.,

$$\begin{aligned} Pr[\text{Type I error}] &= Pr[\text{reject } H_0 \mid H_0 \text{ holds }] \leq \alpha \\ Pr[\text{Type II error}] &= Pr[\text{do not reject } H_0 \mid H_0 \text{ does not hold }] \leq \beta \end{aligned}$$

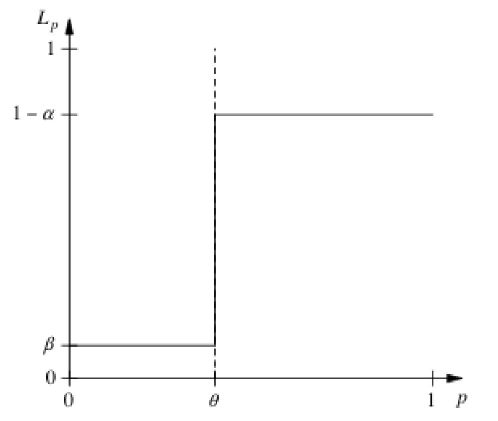
From the above, it can be shown that $Pr[\text{do not reject } H_0 \mid H_0 \text{ holds}] \geq 1 - \alpha$. This implies that the test can correctly distinguish between the two cases $p = r$ and $p = r - \delta$ for some infinitesimally small delta, which in turn, requires that the samples cover the entire model (Younes and Simmons (2002)).

In order to overcome this deficiency, some tolerance (denoted ξ) must be introduced such that the test is not required to distinguish between infinitesimally small probabilities. The new test uses the hypothesis $H_0' : p \geq r + \xi$ and the alternate $H_1' : p \leq r - \xi$. Note that if the new test fails to reject H_0' then it fails to reject H_0. Proceeding as before, let

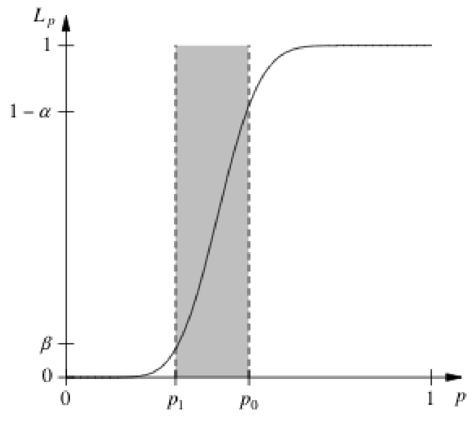
$$\begin{aligned} &Pr[\text{reject } H_0' \mid H_0' \text{ holds }] \leq \alpha \\ &Pr[\text{do not reject } H_0' \mid H_0' \text{ does not hold }] \leq \beta \end{aligned}$$

Therefore, $Pr[\text{do not reject } H_0 \mid H_0' \text{ holds }] \geq 1-\alpha$, and $Pr[\text{do not reject } H_0 \mid H_1' \text{ holds }] \leq \beta$.

Consider p such that $p = r$. Then this new test must fail to reject both H_0' and H_1'. Therefore, this procedure can only be correctly applied for $p \notin [r - \xi, r + \xi]$. This region is



(a) Performance of an ideal testing procedure.



(b) Performance of a testing procedure with an indifference region.

Figure 3.1 Probability (L_p) of accepting the hypothesis $p \geq \theta$.

referred to as the *indifference region*, and 2ξ is called the indifference width. In effect, the probability of rejecting a true hypothesis cannot be bounded in this region. Younes (2005a) provides figures 3.1(a) and 3.1(b) to illustrate the concept of the indifference region.

Younes (2005a) describes several techniques for calculating the appropriate sampling size N, which is a function of α, β and ξ. The simplest is the single sampling plan. For some number of samples n, compute the number of "successful" trials, defined by $C = \sum_{i=1}^{n} X_i$. Then for some pair $\langle c, n \rangle$, if $c \geq \sum_{i=1}^{n} X_i$, reject H_1; otherwise, reject H_0. The challenge is to minimize n so that there exists some integer c such that this procedure will incorrectly reject H_0 with probability at most α, and incorrectly reject H_1 with probability at most β.

Since C is the sum of a series of Bernoulli trials, C has a binomial distribution. The cumulative distribution function for the binomial distribution is given by:

$$F(c; n, p) = Pr(C \leq c) = \sum_{i=1}^{c} \binom{n}{i} p^i (1-p)^{n-i} \tag{3.2}$$

The goal is to select c and n such that

$$\sum_{i=1}^{c} \binom{n}{i} p^i (1-p)^{n-i} \leq \alpha \tag{3.3}$$

and

$$1 - \sum_{i=1}^{c} \binom{n}{i} p^i (1-p)^{n-i} \leq \beta \tag{3.4}$$

There exist infinitely many $\langle c, n \rangle$ pairs that satisfy these inequalities. Since the goal is to minimize the number of samples required, the pair with the smallest n should be selected. In general, there is no simple solution for these equations. Grubbs (1949) provides tables of values for various parameters. Younes (2005a) provides a binary search algorithm that can find these values for arbitrary parameters, and provides an approximation for n given α, β, and ξ. The sample size is inversely proportional to $(2\xi)^2$, and directly proportional to $log(\alpha)$ and $log(\beta)$. Therefore, it is generally more efficient to decrease α and β than to decrease the indifference width.

A minor variation of the single sampling plan stops the testing as soon as c successful or $n - c$ unsuccessful trials have been observed, since the remaining samples cannot change the final result.

The second major technique for determining N is based on the sequential probability ratio test, originally developed by Wald (1945). This test proceeds one sample at a time maintains a count of successful and unsuccessful trials. At each step, two quantities are computed: the probability of the observed sequence given that the true probability is less than $p - \xi$ and the probability of the observed sequence given that the true probability is greather than $p+\xi$. The ratio of these two quantities is compared to two values A and B, which are defined such that $0 < B < A$. If the ratio is less than or equal to B, then H_1 is rejected; if the ratio is greater than or equal to A, then H_0 is rejected. Otherwise, the test proceeds to the next sample.

A and B must be chosen appropriately such that H_0 is incorrectly rejected with probability at most α and H_1 is incorrectly rejected with probability at most β. In practice, Wald (1945) suggests defining $A = \frac{1-\beta}{\alpha}$ and $B = \frac{\beta}{1-\alpha}$, and demonstrates that these values give errors that are at most insignificantly greater than α and β.

The goal of this technique as opposed to the simple sampling plan is to reduce the number of samples required. Younes and Simmons (2006) shows that the expected number of samples is lower for the sequential probability ratio test, but can require a greater number of samples in some cases. In practice, the sequential test is preferable because of the lower expected number of samples.

3.1.2 Implementations of hypothesis testing based approaches

3.1.2.1 `Ymer`

Younes (2005b) describes a tool called `Ymer` that implements this method. Models are described using a variation of the `PRISM` modeling language and properties are expressed in either CSL for continuous time properties, or PCTL for discrete time properties. Younes et al. (2006) note that their method can handle unbounded until properties $\varphi_1 \texttt{ U } \varphi_2$ only when any path in the model either reaches a deadlocked state or a state satisfying $\neg\varphi_1 \vee \varphi_2$. In its original implementation, `Ymer` does not support the unbounded until operator of either CSL or PCTL.

Recently, Younes et al. (2011) proposed two new methods for handling unbounded until properties. The first is based on reachability analysis and requires the construction of the full model before verification can proceed; therefore, it has similar memory requirements to the numerical method. Younes et al. (2011) shows empirically that this method is faster than the numerical method for large models.

The second method is based on a Monte Carlo method for inverting a matrix, developed by von Neumann and Ulam (published by Forsythe and Leibler (1950)). This method has a dependence on the subdominant eigenvalue of the transition matrix, which cannot be computed without building the full model. Younes et al. (2011) simplify the compution the subdominant eigenvalue for parametric models by computing it for small parameters and either using the same value for larger parameters, or adjusting it based on how it should change for the particular model. This approach is not practical for general models because they may not be parametric, or it may not be known how their subdominant eigenvalues scale. As this work is very recent, it is likely that the authors will extend this method to be more general in the future.

3.1.2.2 VESTA

Sen et al. (2005) introduce a new model checking algorithm based on hypothesis testing that can control both Type I and Type II errors. Given a path π and a formula $\psi = \varphi_1 \mathtt{U} \varphi_2$, $\pi \models \psi$ if there exists some finite prefix of π that ends with a state that satisfies φ_2. However, if $\pi \not\models \psi$ there is not necessarily some prefix π that ends with a state that satisfies neither φ_1 nor φ_2. Rather, π may contain states that satisfy only φ_1. In fact, there may be a strongly connected component in the model such that all states in the component satisfy φ_1 but not φ_2. The central observation of Sen et al. (2005) is that for any state s in such a component, $\mathsf{P}(s, \varphi_1 \mathtt{U} \varphi_2) = 0$.

Therefore, this technique defines a basis procedure which verifies $\mathsf{P}(s, \varphi_1 \mathtt{U} \varphi_2) > 0$. For a model $M = (S, s_I, T, L)$, a new model $M' = (S', s_I, T', L')$ is constructed such that:

$$\begin{array}{rcll} S' & = & S \cup \{s_{term}\} & \\ T'\big((t, s_{term})\big) & = & p_s & \forall t \in S \\ T'\big((s_{term}, s_{term})\big) & = & 1 & \\ T'\big((t, u)\big) & = & T(t, u) \cdot (1 - p_s) & \forall t, u \in S \\ L'(s) & = & L(s) & \forall s \in S \\ L'(s_{term}) & = & \emptyset & \end{array}$$

In effect, a new state s_{term} is created such that all other states reach s_{term} with probability p_s, and s_{term} does not satisfy $\varphi_1 \mathtt{U} \varphi_2$. Here p_s is some pre-specified probability that will be discussed in more detail later. In this modified model, for any path π, there exists an extension $\hat{\pi}$ such that $\pi\hat{\pi}$ is a valid path in the model, and the last state in $\hat{\pi}$ does not satisfy $\varphi_1 \mathtt{U} \varphi_2$. That is, starting from any state, there is a non-zero probability of sampling a state that satisfies either φ_2 or $\neg\varphi_1 \wedge \neg\varphi_2$. Note that this same statement cannot be said of the original model M and that this property allows finite samples to be taken from the model M'.

The basis procedure samples paths from the modified model M'. If it ever samples a path that satisfies $\varphi_1 \mathtt{U} \varphi_2$, then $\mathsf{P}(s, \varphi_1 \mathtt{U} \varphi_2) > 0$ for all states in the path (up to the state that satisfies φ_2). If it samples a large number of paths and never finds a path that satisfies $\varphi_1 \mathtt{U} \varphi_2$, then $\mathsf{P}\big(\mathsf{P}(s, \varphi_1 \;\mathtt{U}\; \varphi_2) > 0\big) < \delta$ for some δ.

Unfortunately the basis procedure as described by Sen et al. (2005) is flawed, for two reasons.

First, the basis procedure depends on the parameters p_s and δ_2. The algorithm requires that $p > \frac{\delta_2}{(1-p_s)^N}$, where $p = \mathsf{P}(s, \varphi_1 \;\mathtt{U}\; \varphi_2)$ and N is the number of states in the model. However, since p is the quantity that is desired to be computed, the user cannot appropriately determine a value for δ_2.

Second, Sen et al. (2005) include a proof of the following theorem:

$$p' \geq (1 - p_s)^N \cdot p \tag{3.5}$$

where p' is the probability that a path satisfies the property in M'. This theorem effectively bounds the error from sampling from the modified model instead of sampling from the original model. Younes and Simmons (2006) note that the proof does not hold in general for models with loops.

In He et al. (2010), we propose an alternate to the basis procedure that is valid for any model. The alternate technique requires prior knowledge of the total number of states in the model and its maximum branching factor. The technique is similar to the one proposed by Grosu and Smolka (2005), which deals with sampling based model checking of non-probabilistic models against linear temporal logic (LTL) properties. In contrast to Grosu and Smolka (2005), the technique proposed in this paper does not depend on the state-space of the model and can be used to model check probabilistic properties of probabilistic models.

3.1.2.3 MRMC

Zapreev (2008) proposes a hypothesis testing based statistical technique for verifying unbounded until properties that also uses estimations based on confidence intervals (see Section 9.3 for details). This technique begins by identifying the bottom strongly connected components, or BSCCs, in the model. If a simulation path enters a BSCC and none of the states in the BSCC satisfy $\varphi_1 \;\mathtt{U}\; \varphi_2$, then it is known that the path will never satisfy $\varphi_1 \;\mathtt{U}\; \varphi_2$.

Therefore, sampling for this path can be terminated. The computation of the BSCCs requires preanalysis of the model, and thus limits the application of this technique. The tool `MRMC`, found at MRMC (2010), is based on Zapreev (2008). Katoen and Zapreev (2009) compare `MRMC` with `Ymer` and `VESTA`, and show that both `Ymer` and `VESTA` use less space (constant memory) than `MRMC`, while the verification times of `MRMC` are mostly several factors (up to 10) smaller than those of `Ymer` and `VESTA`. However, the performance of `MRMC` rapidly decreases with growth in model size. A similar and more detailed comparison is presented by Jansen et al. (2007).

3.1.2.4 Ψ^2

For an ergodic `DTMC` with any starting distribution, after some number of steps, the probability distribution at the next state equals the probability distribution at the current state. In effect, the `DTMC` has a steady-state distribution. The number of steps required before the `DTMC` reaches the steady-state distribution is model dependent. Propp and Wilson (1996) describe a procedure that can calculate the appropriate number of steps through simulation. Rabih and Pekergin (2009) propose a new statistical approach to check both steady-state and unbounded until properties utilizing this procedure.

The key insight from Rabih and Pekergin (2009) is that if a sample path reaches the steady-state while always satisfying $\varphi_1 \wedge \neg\varphi_2$, then it will never reach a state that satisfies $\varphi_1 \wedge \varphi_2$. The knowledge of the steady-state distribution effectively allows sampling to be terminated at the appropriate time for any given path.

Rabih and Pekergin (2009) provide a tool called Ψ^2 (available at Rabih and Pekergin (2011)), which implements this method. They provide experimental results from the tool, but no comparison to other tools.

The major drawback of this method is that it requires that the model be ergodic (i.e. irreducible and aperiodic). This condition cannot be guaranteed in general, and determining whether a model is ergodic has space complexity polynomial to the model size and therefore can result in space-space explosion. This condition also precludes the application of this method for models where it is impossible to verify ergodicity, such as black-box systems or other systems where knowledge of the transition structure is not available.

3.1.3 Confidence Interval Based Statistical Estimation

In contrast to the above approaches, these methods attempt first to compute an estimate of the probability p with which paths from s satisfy ψ. Then (if desired), the estimate can be compared to the given bound to determine satisfaction of the property ψ.

Given a state s in a DTMC and a path property ψ, an appropriate number of samples, say N, is obtained starting from s; and $\hat{p}$ (Equation 3.1) is used as the estimate of the probability p with which paths from s satisfy ψ. The estimation method aims to bound the error in the estimate as follows:

$$Pr(|\hat{p} - p| > \epsilon) < \delta$$

In the above, ϵ is the error bound of the estimate and δ is the probability of error in the estimate, i.e., the probability that the estimate is at least ϵ distance away from p. δ is also referred to as the confidence parameter of the estimate. The interval $[p - \epsilon, p + \epsilon]$ is referred to as the confidence interval. Given ϵ and δ, one can use the Chernoff-Hoeffding inequality developed by Hoeffding (1963) to precisely compute the lower bound of the number of samples (N) for probabilistic model checking. Smaller values for ϵ and δ result in larger values of this lower bound.

Herault et al. (2004) have proposed a method to verify a subset of LTL formulae, namely the EPF (Essentially Positive Fragment) in DTMC. Their technique checks whether the probability that a state satisfies an unbounded path property ψ is greater than or equal to r. This is performed using the null hypothesis $H_0 : p \geq r$ against the alternate $H_1 : p < r$. The technique relies on estimating p (Chapter 3.1.3) within a pre-specified error bound and uses the Chernoff-Hoeffding inequality (Hoeffding (1963)) to obtain the appropriate sample size. However, it fails to completely control the error in the procedure.

The sample path length used in the procedure has a pre-specified upper bound. If a simulation reaches that bound and fails to infer a decided result (i.e., whether the path satisfies the given path property or not), the technique assumes that the simulation, if allowed to proceed, eventually will not satisfy the path property under consideration. This assumption allows the method to control Type II error within a pre-specified limit. However, as the authors state in Herault et al. (2004), the proposed technique cannot determine the appropriate upper bound on simulation path length to control the number of the undecided simulations. As such, the method loses control of Type I error (the error that the null hypothesis H_0 holds but the test rejects it).

3.1.4 Implementations of confidence interval based approaches

3.1.4.1 APMC

Herault et al. (2006) describes a tool called APMC, or Approximate Probabilistic Model Checker. As noted above, the tool cannot control error for unbounded path properties. Herault et al. (2004) does not directly address this problem, other than to note that it is an issue.

3.1.4.2 `PRISM`

After the development of APMC, the technique was incorporated in the popular probabilistic model checker `PRISM`, as described by Hinton et al. (2006). The distinguishing feature of `PRISM`'s statistical approach is that unlike the method described above, which allows undecided simulations, `PRISM` will not provide a result to the model checking problem if any sample path remains undecided for some arbitrary, pre-specified length. Although this feature effectively controls Type I error, it becomes necessary to identify a bound on sample length for which the simulation run of each sample will have a decided result. If a sample reaches this predetermined length, the simulation process is aborted and the user is requested to run the experiment again with a larger bound.

All of the results presented in this paper are based on `PRISM` 3.3.1 (released November 22, 2009). Recently, prerelease versions of `PRISM` 4 have become available. This new version of `PRISM` supports both confidence interval and hypothesis testing approaches. More discussion of `PRISM` 4 is included in Chapter 10.2.

3.1.4.3 PRISM-U2B

We propose herein a technique which can be viewed as a natural extension of the technique proposed by Herault et al. (2004) and the algorithm implemented in PRISM. It uses two phases: in the first phase, an appropriate system-dependent bound k_0 in sample length is obtained automatically; and in the second phase, this bound is used to compute the result for unbounded until properties. In effect, a property $\varphi_1 \texttt{ U } \varphi_2$ is converted into the property $\varphi_1 \texttt{ U}^{\leq k_0} \varphi_2$, while maintaining the desired error bounds. To the best of our knowledge, this is the first technique that can estimate the probability of satisfying unbounded until properties in probabilistic DTMC and CTMC models using statistical sampling without any prior knowledge of the model structure to appropriately control the error in estimate.

It should be noted that any of the existing techniques based on either confidence interval estimation or hypothesis testing could be deployed in the second phase of our technique, as the property under consideration is bounded appropriately by k_0, which is computed automatically in the first phase. The results presented here use the technique developed by Herault et al. (2006), as implemented in PRISM.

3.2 Summary

Table 3.2 provides a brief summary and comparison of the model checking tools presented here. The usability rankings are based on Jansen et al. (2007), with $--$ indicating the lowest score, ++ indicating the highest score, and 0 being neutral.

PRISM's modeling language, which is based on the reactive modules formalism developed by Alur and Henzinger (1999), is considered to be the easiest to use. APMC imports PRISM models directly, while MRMC has the option of importing models exported from PRISM. Ymer uses a language similar to PRISM's language.

Ψ^2 is a new tool created by Rabih and Pekergin (2009) to demonstrate the effectiveness of perfect sampling as applied to probabilistic model checking. As such, its modeling and usability scores are low.

Note that most of the tools either do not support unbounded until properties at all, do

Tool	Method	Last update	Until properties		Ease of use[a]	
			Bounded	Unbounded	Modeling	Use
Ymer	Hypothesis	2011	Yes	Yes	$+$	0
Vesta	Hypothesis	2005	Yes	Yes[b]	$--$	$+$
MRMC	Hypothesis	2011	Yes	Yes[c]	$++$	0/+
Ψ^2	Hypothesis	2010	Yes	Yes[d]	$--$	0
APMC	Confidence	2006	Yes	Yes[e]	$++$	$--$
`PRISM` 3	Confidence	2010	Yes	Yes[f]	$++$	$++$
`PRISM` 4	Both	2011	Yes	Yes[f]	$++$	$++$
`PRISM-U2B`	Confidence	2011	Yes	Yes	$++$	$++$

Table 3.1 Comparison of model checking tools

[a]Partially based on Jansen et al. (2007)
[b]Incorrect results for models containing loops
[c]Requires knowledge of model (BSCC detection)
[d]Requires ergodic models
[e]Does not control Type I error
[f]Does not always provide a result

not provide correct results for unbounded until properties, or only support unbounded until properties in certain types of models. Only `PRISM-U2B` provides correct results for unbounded until properties in arbitrary models.

CHAPTER 4 Two-phase Approximate Probabilistic Model Checking

The objective of our work is to reduce unbounded until properties to bounded until properties in the context of probabilistic model checking. The main problem that needs to be addressed to realize such a reduction involves identifying

(a) a suitable bound k_0 for checking the bounded until property (in each simulation) (done in *Phase I*), and

(b) a bound on the number of simulation-paths (each of length k_0) (done in *Phase II*),

such that a statistical sampling based verification result of the bounded until property approximately coincides with that of the unbounded until property within a pre-specified error limit.

4.1 Rationale

The paths belonging to the semantics of $\varphi_1 \; \mathtt{U} \; \varphi_2$ (Section 2.1) can be partitioned into two groups for each $k \geq 1$: one includes the paths that satisfy the property in $\leq k$ steps; while the other includes the paths that satisfy the property in $> k$ steps. I.e., the semantics of $\varphi_1 \; \mathtt{U} \; \varphi_2$ can be written as

$$\begin{aligned}
& \pi \models \varphi_1 \; \mathtt{U} \; \varphi_2 \\
\Leftrightarrow \; & \forall k : \left[\begin{array}{l} \exists 0 \leq i \leq k : \pi[i] \models \varphi_2 \;\wedge\; \forall j < i : \pi[j] \models \varphi_1 \\ \bigvee \\ \exists i > k : \pi[i] \models \varphi_2 \;\wedge\; \forall j < i : \pi[j] \models \varphi_1 \wedge \neg\varphi_2 \end{array} \right] \\
\Leftrightarrow \; & \pi \models \forall k : \left[(\varphi_1 \; \mathtt{U}^{\leq k} \; \varphi_2) \;\vee\; (\varphi_1 \; \mathtt{U}^{>k} \; \varphi_2) \right]
\end{aligned}$$

Note that we have defined that $\pi \models \varphi_1 \, \mathtt{U}^{>k} \, \varphi_2$ if and only if the first state $\pi[i]$ that satisfies φ_2 appears in π after at least $k+1$ steps and φ_1 is satisfied in all states before $\pi[i]$.

Since $(\varphi_1 \, \mathtt{U}^{\leq k} \, \varphi_2) \ \wedge \ (\varphi_1 \, \mathtt{U}^{>k} \, \varphi_2) = \mathit{ff}$, by law of total probability

$$\mathsf{P}(s, \varphi_1 \, \mathtt{U} \, \varphi_2) = \mathsf{P}(s, \varphi_1 \, \mathtt{U}^{\leq k} \, \varphi_2) \ + \ \mathsf{P}(s, \varphi_1 \, \mathtt{U}^{>k} \, \varphi_2) \tag{4.1}$$

From the fact that probabilities $\in [0,1]$,

$$0 \leq \mathsf{P}(s, \varphi_1 \, \mathtt{U} \, \varphi_2) - \mathsf{P}(s, \varphi_1 \, \mathtt{U}^{\leq k} \, \varphi_2) \ = \ \mathsf{P}(s, \varphi_1 \, \mathtt{U}^{>k} \, \varphi_2) \tag{4.2}$$

Next consider the property $\varphi_1 \, \mathtt{U}^{>k} \, \varphi_2$.

$$\begin{aligned}
& \pi \models \varphi_1 \, \mathtt{U}^{>k} \, \varphi_2 \\
\Leftrightarrow \quad & \exists i > k : \pi[i] \models \varphi_2 \ \wedge \ \forall j < i : \pi[j] \models \varphi_1 \ \wedge \ \neg\varphi_2 \\
\Rightarrow \quad & \forall i \leq k : \pi[i] \models \varphi_1 \wedge \neg\varphi_2 \\
\Leftrightarrow \quad & \varphi_1 \, \mathtt{U} \, \varphi_2 \text{ is neither satisfied nor unsatisfied in } k \text{ steps from } \pi[0] \\
\Leftrightarrow \quad & \pi \models \neg(\varphi_1 \, \mathtt{U}^{\leq k} \, \varphi_2) \ \wedge \ \neg(\neg\varphi_2 \, \mathtt{U}^{\leq k} \, (\neg\varphi_1 \wedge \neg\varphi_2))
\end{aligned}$$

Let $\psi_k = (\varphi_1 \, \mathtt{U}^{\leq k} \, \varphi_2) \ \vee \ (\neg\varphi_2 \, \mathtt{U}^{\leq k} \, (\neg\varphi_1 \wedge \neg\varphi_2))$, i.e., ψ_k is the property that is satisfied by a path π only when the satisfiability of $\varphi_1 \, \mathtt{U} \, \varphi_2$ can be proved or disproved in k steps from the start state ($\pi[0]$). Therefore, from the above, $(\varphi_1 \, \mathtt{U}^{>k} \, \varphi_2) \Rightarrow \neg\psi_k$ and

$$\mathsf{P}(s, \varphi_1 \, \mathtt{U}^{>k} \, \varphi_2) \leq \mathsf{P}(s, \neg\psi_k) = 1 - \mathsf{P}(s, \psi_k) \tag{4.3}$$

From Equations 4.2 and 4.3, for any $k \geq 1$ we obtain

$$0 \leq \mathsf{P}(s, \varphi_1 \, \mathtt{U} \, \varphi_2) - \mathsf{P}(s, \varphi_1 \, \mathtt{U}^{\leq k} \, \varphi_2) \leq 1 - \mathsf{P}(s, \psi_k) \tag{4.4}$$

Our objective is to select a k_0 such that for any given ϵ_0.

$$\mathsf{P}(s, \psi_{k_0}) \geq 1 - \epsilon_0 \tag{4.5}$$

In that case,

$$0 \ \leq \ \mathsf{P}(s, \varphi_1 \, \mathtt{U} \, \varphi_2) - \mathsf{P}(s, \varphi_1 \, \mathtt{U}^{\leq k_0} \, \varphi_2) \ \leq \ 1 - \mathsf{P}(s, \psi_{k_0}) \leq \epsilon_0 \tag{4.6}$$

In other words, by choosing an appropriate k_0, the probability of satisfying the unbounded path property $\varphi_1 \, \mathtt{U} \, \varphi_2$ can be made close (within an error margin of ϵ_0, for any arbitrarily small choice of ϵ_0) to the probability of satisfying the bounded path property $\varphi_1 \, \mathtt{U}^{\leq k_0} \, \varphi_2$.

4.2 U2B: Two-phase Approximate Model Checking for DTMC

The discussion in the previous section (specifically, Equations 4.5 and 4.6) motivates our two phase method. In the first phase we determine a suitable k_0 and in the second phase we estimate $\mathsf{P}(s, \varphi_1 \;\mathtt{U}^{\leq k_0}\; \varphi_2)$. Finally, we use this estimate of $\mathsf{P}(s, \varphi_1 \;\mathtt{U}^{\leq k_0}\; \varphi_2)$ as the estimate of $\mathsf{P}(s, \varphi_1 \;\mathtt{U}\; \varphi_2)$. Our method utilizes confidence interval based statistical estimation (see Section 3.1.3) in both phases.

In *Phase I*, $\mathsf{P}(s, \psi_k)$ is estimated for different values of k using N_1 Monte Carlo simulation paths similar to the GAA (Generic Approximation Algorithm) described in Herault et al. (2004). This is done for all $k \geq 1$ until for some k_0, the estimate satisfies Equation 4.5.

Once k_0 is obtained, in *Phase II* we estimate $\mathsf{P}(s, \varphi_1 \;\mathtt{U}^{\leq k_0}\; \varphi_2)$. This estimate is computed as the proportion of N_2 Monte Carlo simulation paths (each of length at most k_0) that satisfy $\varphi_1 \;\mathtt{U}^{\leq k_0}\; \varphi_2$. This also can be thought of as a simple application of the GAA algorithm for bounded until properties described in Herault et al. (2004).

The two phases for computing k_0 and then computing $\mathsf{P}(s, \varphi_1 \;\mathtt{U}^{\leq k_0}\; \varphi_2)$ are carried out "independently", i.e., involving separate samples (of sizes N_1 and N_2 respectively), which enables us to combine the errors in two phases to guarantee a certain precision. The number of Monte Carlo simulation paths used in the two phases and the value of k_0 are chosen in a way that controls the errors in each of the phases and combines them to guarantee the correctness of the final estimate within a pre-specified error limit (see Theorem 1 below). In short, our method, $\mathtt{U2B}(M, s, \varphi_1 \;\mathtt{U}\; \varphi_2, \epsilon, \delta)$, takes as input the DTMC model M, the state s, the until property under consideration, the error margin ϵ and the confidence parameter δ; and returns the estimate of $\mathsf{P}(s, \varphi_1 \;\mathtt{U}\; \varphi_2)$ within ϵ bound with a high degree of certainty (at least $1-\delta$). The steps of our method are summarized as follows:

Main Steps. $\mathtt{U2B}(M, s, \varphi_1 \;\mathtt{U}\; \varphi_2, \epsilon, \delta)$

1. *Phase I*: Obtaining k_0

   (a) Choose $N_1 \geq N_1^* = 9\log(\frac{4}{\delta})/2\epsilon^2$. From M, obtain N_1 Monte Carlo simulation paths of length $k = 1$. For $i = 1, \ldots, N_1$, let $X_i = 1$ if the i-th simulation satisfies ψ_k;

$X_i = 0$ otherwise.

(b) Estimate $\mathsf{P}(s, \psi_k)$ as the proportion of the simulation paths satisfying ψ_k, i.e.

$$\widehat{\mathsf{P}}(s, \psi_k) = \frac{1}{N_1} \sum_{i=1}^{N_1} X_i. \tag{4.7}$$

(c) Verify if Equation 4.5 is satisfied by the estimate in Equation 4.7 with the current value of k and $\epsilon_0 = \frac{\epsilon}{3}$. More precisely, if

$$\widehat{\mathsf{P}}(s, \psi_k) \geq 1 - \epsilon_0 = 1 - \frac{\epsilon}{3}, \tag{4.8}$$

then $k_0 = k$ and proceed to *Phase II*. Otherwise, increase k by 1 and generate one more transition for each of the existing N_1 simulation paths, creating N_1 paths of increased (by 1) length. Define $X_i, i = 1, \ldots, N_1$ as in Step 1(a) using these extended simulation paths and repeat the Step 1(b)-(c).

2. *Phase II*: Estimating $\mathsf{P}(s, \varphi_1 \ \mathtt{U}^{\leq k_0} \ \varphi_2)$

(a) Choose $N_2 \geq N_1^* = 36 \log(\frac{4}{\delta})/\epsilon^2$. From M, obtain N_2 Monte Carlo simulation paths (of length at most k_0). For $i = 1, \ldots, N_2$, let $Y_i = 1$ if the i-th simulation path satisfies $\varphi_1 \ \mathtt{U}^{\leq k_0} \ \varphi_2$; $Y_i = 0$ otherwise.

(b) Estimate $\mathsf{P}(s, \varphi_1 \ \mathtt{U}^{\leq k_0} \ \varphi_2)$ as the proportion of the simulation paths that satisfy $\varphi_1 \ \mathtt{U}^{\leq k_0} \ \varphi_2$, i.e

$$\widehat{\mathsf{P}}(s, \varphi_1 \ \mathtt{U}^{\leq k_0} \ \varphi_2) = \frac{1}{N_2} \sum_{i=1}^{N_2} Y_i. \tag{4.9}$$

Return $\widehat{\mathsf{P}}(s, \varphi_1 \ \mathtt{U}^{\leq k_0} \ \varphi_2)$, as the estimate for $\mathsf{P}(s, \varphi_1 \ \mathtt{U} \ \varphi_2)$

4.3 Proof of Correctness

The following theorem states the correctness of our method.

Theorem 1. *Given any precision parameter $\epsilon > 0$ and confidence parameter $\delta > 0$, the estimator $\mathtt{U2B}(M, s, \varphi_1 \ \mathtt{U} \ \varphi_2, \epsilon, \delta)$ with the chosen values of k_0, N_1^*, N_2^* satisfies the following:*

$$Pr\left(\mid \mathtt{U2B}(M, s, \varphi_1 \ \mathtt{U} \ \varphi_2, \epsilon, \delta) - \mathsf{P}(s, \varphi_1 \ \mathtt{U} \ \varphi_2) \mid > \ \epsilon\right) \ \leq \ \delta. \tag{4.10}$$

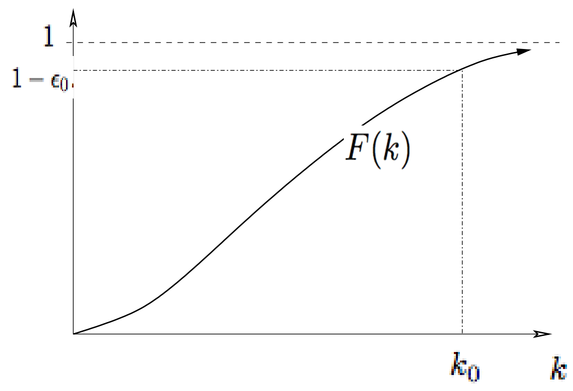


Figure 4.1 Choosing k_0 in *Phase I* from $F(k)$

We begin by discussing auxiliary results in theoretical statistics that will be used in proving the above theorem. We discuss properties of the estimation procedure separately for the two phases.

4.3.1 Phase I: Estimating k_0

Let $F(\cdot)$ be a function where $F(k) = \mathsf{P}(s, \psi_k)$, $k \geq 1$. Figure 4.1 illustrates the sample valuations of $F(\cdot)$ for different valuations of k; we refer to this graph as the *Decided Graph* (D-Graph); $F(k)$ being the probability that $\varphi_1 \mathtt{U} \varphi_2$ is *decided* in $\leq k$ steps from the state s. The main challenge in *Phase I* is that the function $F(\cdot)$ is not typically known. If this function were known, finding a k_0 that satisfies Equation 4.5 could have been achieved by simply inverting this (non-decreasing) function.

The function $F(\cdot)$ can be thought of as the cumulative distribution function (c.d.f) of a random variable $K =$ the minimum number of transitions required to verify $\varphi_1 \mathtt{U} \varphi_2$ along a randomly selected simulation path in the given model. In this way, our estimation process in *Phase I* is equivalent to estimating this c.d.f using N_1 independent samples collected from the distribution of this variable K. In fact, our estimate $\widehat{\mathsf{P}}(s, \psi_k)$ (as a function of k) is the usual empirical c.d.f estimator $\hat{F}_{N_1}(\cdot)$ of the true c.d.f $F(\cdot)$. It is well known that $k \geq 1$, $\hat{F}_{N_1}(k)$ converges to $F(k)$, as $N_1 \to \infty$ at a suitable rate, for *each* k. For the proof of Theorem 1, we need the rate *uniform* in k at which this convergence takes place. This is provided by the celebrated Dvoretzky-Kiefer-Wolfowitz (DKW) inequality (see for example, Massart (1990)): For each $\epsilon_1 > 0, N_1 \geq 1$, $Pr\left(\sup_{k\geq 1} |\hat{F}_{N_1}(k) - F(k)| > \epsilon_1\right) \leq 2e^{-2N_1(\epsilon_1)^2}$. This result, restated in terms of $\mathsf{P}(s, \psi_k) = F(k)$ and $\widehat{\mathsf{P}}(s, \psi_k) = \hat{F}_{N_1}(k)$, for each k, yields the following lemma, which will be needed for our proof of Theorem 1.

Lemma 1. *Given any $\epsilon_1 > 0$ and $N_1 \geq 1$,*

$$Pr\left(\sup_{k\geq 1} | \widehat{\mathsf{P}}(s, \psi_k) - \mathsf{P}(s, \psi_k) | > \epsilon_1\right) \leq 2e^{-2N_1(\epsilon_1)^2}.$$

4.3.2 Phase II: Estimating $\mathsf{P}(s, \varphi_1 \ \mathtt{U}^{\leq k_0} \ \varphi_2)$

In this phase, we estimate the probability of the *bounded* until property $\varphi_1 \ \mathtt{U}^{\leq k_0} \ \varphi_2$ in M, with $k_0 \geq 1$ as determined in *Phase I*. For any given $k \geq 1$, our algorithm in *Phase II* is simply the GAA algorithm (c.f. Herault et al. (2004)) of estimating the probability of a *bounded* until property $\varphi_1 \ \mathtt{U}^{\leq k} \ \varphi_2$ in M. Hence using the same technique (i.e., using Chernoff-Hoeffding bound) we get for each $\epsilon_2 > 0$,

$$Pr\left(| \widehat{\mathsf{P}}(s, \varphi_1 \ \mathtt{U}^{\leq k} \ \varphi_2) - \mathsf{P}(s, \varphi_1 \ \mathtt{U}^{\leq k} \ \varphi_2) | > \epsilon_2\right) \leq 2e^{-N_2(\epsilon_2)^2/4}.$$

Now since the above inequality is true for all $k \geq 1$, it is true *conditional* on the simulations of *Phase I*, for $k = k_0$. But the two phases are carried out independently, which means the above statement must be true *unconditionally* as well, for $k = k_0$. Summarizing this discussion, we have

Lemma 2. *Given any $\epsilon_2 > 0$ and $N_2 \geq 1$, for k_0 given by* Phase I *of U2B*

$$Pr\left(| \widehat{P}(s, \varphi_1 \ U^{\leq k_0} \ \varphi_2) - P(s, \varphi_1 \ U^{\leq k_0} \ \varphi_2) | > \epsilon_2\right) \leq 2e^{-N_2(\epsilon_2)^2/4}.$$

Now we use the results of Lemmas 1 and 2 to complete the proof of Theorem 1.

Proof of Theorem 1. The triangle inequality (after adding and subtracting suitable terms) yields the following

$$\begin{aligned} | \mathtt{U2B}(M, s, \varphi_1 \ \mathtt{U} \ \varphi_2, \epsilon, \delta) - \mathsf{P}(s, \varphi_1 \ \mathtt{U} \ \varphi_2) | &= | \widehat{\mathsf{P}}(s, \varphi_1 \ \mathtt{U}^{\leq k_0} \ \varphi_2) - \mathsf{P}(s, \varphi_1 \ \mathtt{U} \ \varphi_2) | \\ \leq \ | \widehat{\mathsf{P}}(s, \varphi_1 \ \mathtt{U}^{\leq k_0} \ \varphi_2) \ - \ \mathsf{P}(s, \varphi_1 \ \mathtt{U}^{\leq k_0} \ \varphi_2) | & \\ + \ | \mathsf{P}(s, \varphi_1 \ \mathtt{U} \ \varphi_2) - \mathsf{P}(s, \varphi_1 \ \mathtt{U}^{\leq k_0} \ \varphi_2) | \ . & \qquad (4.11) \end{aligned}$$

Recall from Equation 4.8 that we have $1 - \widehat{\mathsf{P}}(s, \psi_k) \leq \epsilon/3$. Hence, using Equation 4.4 and the triangle inequality, we get the following bound on the last term in Equation 4.11

$$\begin{aligned} \mid \mathsf{P}(s, \varphi_1 \mathtt{U}\ \varphi_2) - \mathsf{P}(s, \varphi_1 \mathtt{U}^{\leq k_0}\ \varphi_2) \mid &\leq (1 - \mathsf{P}(s, \psi_{k_0})) \\ &\leq (1 - \widehat{\mathsf{P}}(s, \psi_{k_0})) + \mid \widehat{\mathsf{P}}(s, \psi_{k_0}) - \mathsf{P}(s, \psi_{k_0}) \mid \\ &\leq \frac{\epsilon}{3} + \sup_{k \geq 1} \mid \widehat{\mathsf{P}}(s, \psi_k) - \mathsf{P}(s, \psi_k) \mid . \end{aligned} \tag{4.12}$$

Combining Equation 4.11 with Equation 4.12, we get the following bound

$$\begin{aligned} &\mid \mathtt{U2B}(M, s, \varphi_1 \mathtt{U}\ \varphi_2, \epsilon, \delta) - \mathsf{P}(s, \varphi_1 \mathtt{U}\ \varphi_2) \mid \leq \\ &\mid \widehat{\mathsf{P}}(s, \varphi_1 \mathtt{U}^{\leq k_0}\ \varphi_2) - \mathsf{P}(s, \varphi_1 \mathtt{U}^{\leq k_0}\ \varphi_2) \mid\ + \tfrac{\epsilon}{3}\ +\ \sup_{k \geq 1} \mid \widehat{\mathsf{P}}(s, \psi_k) - \mathsf{P}(s, \psi_k) \mid \end{aligned}$$

Hence, the fact that the left side of the above inequality is greater than ϵ (see Equation 4.10 in Theorem 1) implies that at least one of the terms on the right side is greater than $\epsilon/3$. Therefore, we obtain the following:

$$\begin{aligned} &Pr\left(\mid \mathtt{U2B}(M, s, \varphi_1 \mathtt{U}\ \varphi_2, \epsilon, \delta) - \mathsf{P}(s, \varphi_1 \mathtt{U}\ \varphi_2) \mid > \epsilon\right) \\ &\quad \leq \quad Pr\left(\mid \widehat{\mathsf{P}}(s, \varphi_1 \mathtt{U}^{\leq k_0}\ \varphi_2)\ -\ \mathsf{P}(s, \varphi_1 \mathtt{U}^{\leq k_0}\ \varphi_2) \mid > \frac{\epsilon}{3}\right) \\ &\qquad\qquad + Pr\left(\sup_{k \geq 1} \mid \widehat{\mathsf{P}}(s, \psi_k) - \mathsf{P}(s, \psi_k) \mid > \frac{\epsilon}{3}\right) \quad \leq \quad \delta. \end{aligned} \tag{4.13}$$

The last inequality follows from the bounds in Lemmas 1 and 2 with $\epsilon_1 = \epsilon/3$ and $\epsilon_2 = \epsilon/3$, since with $N_i \geq N_i^*, i = 1, 2$, we have $2e^{-2N_1(\epsilon_1)^2} \leq \delta/2$ (i.e, $N_1 \geq \frac{1}{2\epsilon_1^2} log(\frac{4}{\delta}) \geq \frac{9}{2\epsilon^2} log(\frac{4}{\delta})$) and $2e^{-N_2(\epsilon_2)^2/4} \leq \delta/2$ (i.e., $N_2 \geq \frac{4}{\epsilon_2^2} log(\frac{4}{\delta}) \geq \frac{36}{\epsilon^2} log(\frac{4}{\delta})$). This completes the proof of Theorem 1.

□

Remark 1. *Observe that there are three error bounds that are derived from ϵ, each of which is assigned to $\frac{\epsilon}{3}$: ϵ_0 (From Equation 4.8), ϵ_1 (From Lemma 1) and ϵ_2 (From Lemma 2). While ϵ_0 is the measure of closeness of $\widehat{\mathsf{P}}(s, \psi_k)$ to 1, ϵ_1 and ϵ_2 capture the closeness of the estimated and true probabilities in each phase. In other words, a smaller ϵ_0 will lead to a larger value for k_0 (sample path length), while smaller ϵ_1 and ϵ_2 values will result in larger values for N_1 and N_2, the sample sizes in each phase. The proof of our theorem holds as long as $\epsilon_0 + \epsilon_1 + \epsilon_2 = \epsilon$. These values can be fine tuned experimentally.*

CHAPTER 5 Efficient & Practical Realization of `U2B` Model Checker

As discussed in Section 4.2, our method, `U2B` involves two phases. In this chapter, we present a direct (naive) implementation of *Phase I* followed by an optimized variation. Recall that *Phase II* deals with estimating $\mathsf{P}(s, \varphi_1 \, \mathtt{U}^{\leq k_0} \, \varphi_2)$, which can be immediately computed using the existing statistical method present in `PRISM`; as such, we do not provide any additional optimization for *Phase II*. The integration of this method into `PRISM` (i.e. the development of `PRISM-U2B`) is discussed in Chapter 8.

5.1 Naive Implementation of Phase I

Algorithm 1 lists the procedure IDENTIFYK0, describing a naive realization of *Phase I* in `U2B`. `workingSet` contains the set of current states (last states of simulation paths) that are examined in *Phase I*. It is initialized with N_1 copies of the start state of the model; these are the current states in N_1 paths of length 0 (Line 3). The variable `trueCount`, initialized to 0 (Line 4), captures the number of paths in `workingSet` that satisfy ψ_k.

The states in `workingSet` are verified to check whether ψ_k is satisfied; if satisfied, the states are removed from `workingSet` and `trueCount` is incremented (Lines 6–9). Observe that `trueCount` captures the number of paths of length k that satisfy ψ_k. Therefore, `trueCount`$/N_1$ computes $\widehat{\mathsf{P}}(s, \psi_k)$ (Equation 4.7). The loop (Lines 5) is terminated if $N_1(1 - \epsilon_0) \leq$ `trueCount` (Lines 10, 11), i.e., if the proportion of paths in the sample of size N_1 that satisfy ψ_k is less than $1 - \epsilon_0$.

If the terminating condition is not satisfied, k is incremented and `workingSet` is updated by obtaining the next states of the paths in the working set by random selection based on the probability distribution in the model. On termination of the iteration, i.e., when

Algorithm 1 Naive Implementation of *Phase I*

```
1:  procedure IDENTIFYK0($N_1, \epsilon_0, \psi$)
                  ▷ $N_1$: Total number of samples for Phase I (see U2B Step 1(a))
                  ▷ $\epsilon_0$ closeness to 1 (see U2B Step 1(c))
                  ▷ $\psi := (\varphi_1$ U $\varphi_2) \vee (\neg\varphi_2$ U $\neg\varphi_1 \wedge \neg\varphi_2)$
2:      $k := 0$;
3:      workingSet := add $N_1$ copies of start state;        ▷ path length $k = 0$
4:      trueCount := 0;        ▷ Initial number of paths that satisfy $\psi_k$
5:      while True do
6:          for each state ∈ workingSet that satisfies $\neg\varphi_1 \vee \varphi_2$ do        ▷ i.e., path satisfies $\psi_k$
7:              remove state from workingSet;
8:              trueCount++;
9:          end for
10:         if ($N_1(1 - \epsilon_0) \leq$ trueCount) then
11:             return $k$;        ▷ trueCount $\geq N_1(1-\epsilon_0)$, i.e., $\widehat{\mathsf{P}}(s,\psi_k) = \frac{\texttt{truecount}}{N_1} \geq 1-\epsilon_0$
12:         end if
13:         $k$++;
14:         replace current states in workingSet with randomly obtained next states;
15:     end while
16: end procedure
```

trueCount$/N_1 \geq 1 - \epsilon_0$, the value k is returned. This concludes *Phase I* of the U2B and initiates the invocation of *Phase II*.

Remark 2. *Algorithm 1 terminates and returns k if and only if $\widehat{\mathsf{P}}(s,\psi_k) \geq 1-\epsilon_0$. The above statement holds directly from the facts that $\widehat{\mathsf{P}}(s,\psi_k)$ is equal to the proportion of paths in sample of size N_1 that satisfy ψ_k, and that Algorithm 1 considers a workingSet of N_1 paths and terminates only when trueCount $\geq N_1(1-\epsilon_0)$.*

5.2 Optimization: Reducing the workingSet in Phase I

Algorithm 1 initializes workingSet to a set (of size N_1) of paths of length 0, and maintains a subset of paths that do not satisfy ψ_k until the loop terminating condition at Line 5 is satisfied. This can be expensive in terms of space usage because N_1 is often large.

An alternative realization of *Phase I* has been developed in which only a small subset of the sample (of total size N_1) is considered in workingSet. This is achieved from the observation that on termination of *Phase I*, the number of paths that do not satisfy ψ_k can be at most $N_1\epsilon_0 - 1$ in the sample of size N_1 (as trueCount$/N_1$ is required to be $\geq 1-\epsilon_0$). Therefore, it is sufficient to consider only $N_1\epsilon_0$ paths in workingSet at any point in time. Whenever

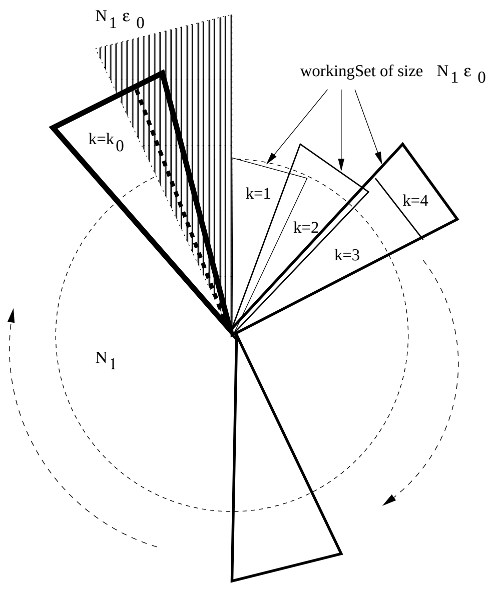


Figure 5.1 Optimized realization of *Phase I* of `U2B`, `workingSet` of size $N_1\epsilon_0$: Sliding window strategy.

some paths in `workingSet` satisfy ψ_k, they are replaced with a new set of paths such that the size of `workingSet` is maintained at $N_1\epsilon_0$. These new paths are extended to length k before k is incremented and the old paths are extended. In short, a sliding window `workingSet` is considered until the total number of paths that satisfy ψ_k is $\geq N_1(1-\epsilon_0)$ (as required in *Phase* I). This reduces the space usage as the number of paths stored in `workingSet` at any time is fixed to $N_1\epsilon_0$, a small proportion of N_1.

Figure 5.1 illustrates this strategy. Each sector represents the working set of paths examined at each iteration. The working set size is fixed to $N_1\epsilon_0$. The radius of each sector denotes the length of paths in the corresponding working set. In the figure, initially a working set containing paths of length 1 is considered. The property ψ_k is satisfied in some of these paths and as a result, those paths are removed and new paths of length 1 are generated and added to the working set. All paths in the working set are then extended by one step. This is repeated in iteration 2, at the end of which the working set contains paths of length 3. As per the figure, none of the paths in the working set satisfy ψ_k and as such, no new paths are added to the working set; instead all paths in the working set are extended by one step (i.e., the path length k becomes 4). The above process is repeated until a sector (working set) is obtained such that (a) it overlaps with the shaded sector and (b) contains a sub-sector of paths (denoted by dotted lines) that satisfy ψ_k and that also overlaps with the shaded sector. This implies that the proportion of paths that satisfy ψ_k is $\geq 1 - \epsilon_0$. Algorithm 2 presents the listing of this optimized implementation.

Algorithm 2 Optimized Implementation of *Phase I*

```
1:  procedure IDENTIFYK0_OPT_1($N_1, \epsilon_o, \psi$)
                          ▷ $N_1$: Total number of samples for Phase I (see U2B Step 1(a))
                          ▷ $\epsilon_0$: Error bound for Phase I (see U2B Step 1(c))
                          ▷ $\psi := (\varphi_1$ U $\varphi_2) \vee (\neg\varphi_2$ U $\neg\varphi_1 \wedge \neg\varphi_2)$
2:      $k := 0$;                                              ▷ Initial length of paths
3:      $\boxed{\texttt{workingSet} := \text{add } N_1\epsilon_0 \text{ copies of start state;}}$    ▷ paths of length $k = 0$
4:      trueCount := 0;                ▷ Initial number of paths that satisfy $\psi_k$
5:      while True do
6:          for each state ∈ workingSet that satisfies $\neg\varphi_1 \vee \varphi_2$ do    ▷ i.e., path satisfies $\psi_k$
7:              remove state from workingSet;
8:              trueCount++;
9:          end for
10:         if ($N_1$(1 - $\epsilon_0$) ≤ trueCount) then
11:             return $k$;      ▷ trueCount $\geq N_1(1-\epsilon_0)$, i.e., $\widehat{\mathsf{P}}(s,\psi_k) = \frac{\texttt{truecount}}{N_1} \geq 1-\epsilon_0$
12:         end if
13:         $k$++;
14:         replace current states in workingSet with randomly obtained next states;
15:         $\boxed{\text{run } (N_1\epsilon_0 \text{ - } |\texttt{workingSet}|) \text{ random simulations for } k \text{ steps from start state;}}$
16:         $\boxed{\text{add last states to } \texttt{workingSet}\text{;}}$
17:     end while
18: end procedure
```

The primary differences between Algorithms 1 and 2 are at Lines 3 and 15–16 of Algorithm 2. Instead of initializing `workingSet` with N_1 copies of start states (as in Algorithm 1), it is initialized with $N_1\epsilon_0$ copies of start states. Then, if some states are removed from `workingSet` (Line 7), it is replenished with the last states of newly extended paths of appropriate length such that the total number of states in `workingSet` remains equal to $N_1\epsilon_0$ (Line 15, 16).

Proposition 1. *Algorithm 2 terminates and returns k if and only if $\widehat{\mathsf{P}}(s,\psi_k) \geq 1-\epsilon_0$.*

Algorithm 2 terminates if $N_1(1-\epsilon_0) \leq$ `trueCount`. Therefore, it must examine at least $N_1(1-\epsilon_0)+1$ paths before terminating and returning a k. That is, at most $N_1\epsilon_0 - 1$ paths are not considered by Algorithm 2 for returning a k and inferring that $\widehat{\mathsf{P}}(s,\psi_k) \geq 1-\epsilon_0$.

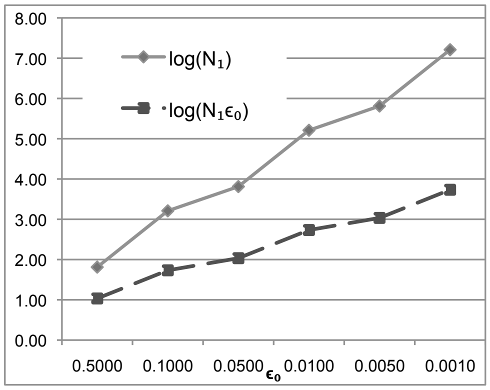


Figure 5.2 Logarithms of N_1 and $N_1\epsilon_0$ against ϵ (x-axis).

The above proposition holds as $\widehat{\mathsf{P}}(s, \psi_k)$ is the proportion of paths (i.e., `trueCount`) in a sample of size N_1 that satisfy ψ_k and the terminating condition of Algorithm 2 ensures that `trueCount`$/N_1 \geq 1 - \epsilon_0$.

5.2.1 Discussion: Algorithm 1 Vs. Algorithm 2

Recall that Algorithm 1 uses $N_1 = \frac{1}{2\epsilon_1^2} log(\frac{4}{\delta})$ samples while Algorithm 2 requires $N_1\epsilon_0$ samples. We have considered $\epsilon_0 = \epsilon_1 = \frac{\epsilon}{3}$ (see Remark 1 in Section 4.3); however any other choice of ϵ_0 and ϵ_1 as a function of ϵ would suffice for this discussion. Figure 5.2 compares the increases (in log scale) in the value of N_1 and $N_1\epsilon_0$ with the decrease in ϵ for a specific confidence parameter $\delta = 0.001$. Observe that N_1 increases *faster* than $N_1\epsilon_0$ with the decrease in error bound ϵ of the `U2B` method (for instance, with $\epsilon = 0.001$, $N_1 = 1.6 \times 10^7$ while $N_1\epsilon_0 = 5403$). In other words, as the precision of the `U2B` method is increased, Algorithm 2 uses a considerably smaller working set (and in turn, less space) compared to that used by Algorithm 1; the larger the precision, the greater the benefit in terms of space usage in using Algorithm 2 over Algorithm 1.

The space-saving benefit increases with the increase in the number of variables describing the model (i.e., each model state). We have experimented with different variations of a simple queue model (see Jennings et al. (2010)) where each variation contains a different number of variables. Figure 5.3 shows that with the increase in the number of variables in the model, the

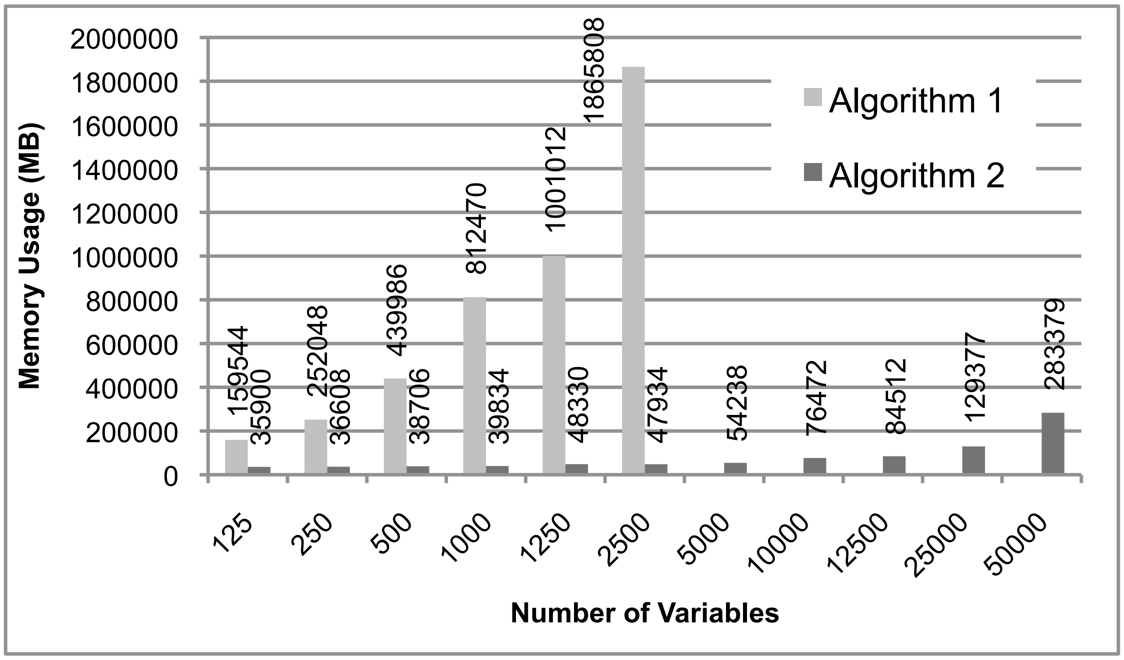


Figure 5.3 Memory usage by Algorithm 1 Vs. Algorithm 2 for variations of a simple queue model.

increase in the space usage (in MB) by Algorithm 1 is larger than that in Algorithm 2. In fact, when the number of variables is ≥ 2500, Algorithm 1 reports out of memory and terminates without computing any result.

CHAPTER 6 Dealing with Non-termination in `U2B`

Recall that $F(k)$ is the probability that $\varphi_1 \ \mathtt{U} \ \varphi_2$ is decided in k steps from a state s under consideration. In Section 4.3.1 we introduced the notion of the Decided Graph (D-Graph; Figure 4.1) presenting the valuations of $F(\cdot)$ for different valuations of k. In this chapter, we discuss the condition on the valuation of $F(\cdot)$ (and the corresponding shape of the D-Graph) under which the `U2B` method fails to terminate and propose a heuristic-based alternate method to ensure termination.

Our argument for the applicability of the `U2B` method requires that as $k \to \infty$, the limit of $F(k) = \mathsf{P}(s, \psi_k) \geq 1 - \epsilon_0$ (see Equation 4.5). That is,

$$\lim_{k \to \infty} \mathsf{P}(s, \psi_k) \geq 1 - \epsilon_0 \tag{6.1}$$

When the limit is 1, Equation 6.1 is satisfied for any ϵ_0 and hence our method is valid for any $\epsilon_0 \geq 0$. (See Figure 4.1, which illustrates the case when the limit is 1.)

When the limit is not equal to 1, our method works (i.e., *Phase I* computation terminates) for any ϵ_0 that satisfies the inequality in Equation 6.1. See Figure 6.1 for an example D-Graph from such a model. There are several local plateaus, followed by an unbounded plateau. Note that the requirement of satisfying this inequality does not restrict the applicability of our technique to any specific class of properties or models; our technique can be applied for any until property and for any model as long as the above requirement is satisfied. It is worth mentioning that `PRISM`'s statistical method is equivalent to choosing $\epsilon_0 = 0$ in our method and hence will fail to provide a result whenever this limit is not equal to 1. In other words, whenever `PRISM`'s statistical verification method is successful in computing a result, our method also terminates with a result; and furthermore, our method is able to estimate probabilities for many cases where `PRISM`'s statistical method fails (see discussion in Section 9).

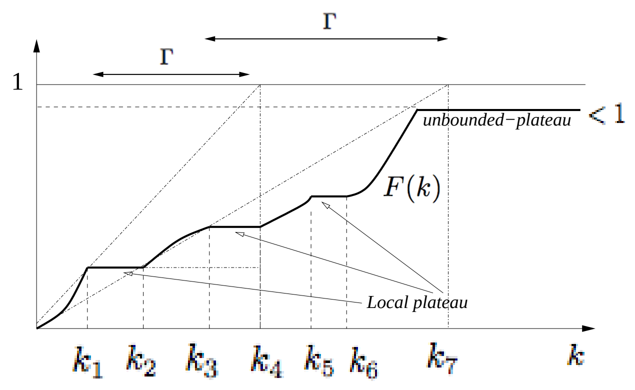


Figure 6.1 Plateaus in the D-Graph.

The implication of the limit not being equal to 1 is that $\mathsf{P}(s, \neg(tt \; \mathtt{U} \; (\varphi_2 \vee \neg\varphi_1))) > 0$, which happens if and only if there exists a path to a strongly connected component in the model where every state satisfies $(\varphi_1 \wedge \neg\varphi_2)$. If the probability of such a path (or paths) is greater than ϵ_0 then the *Phase I* computation will fail to terminate. It is worth mentioning that in such a case, any statistical verification method that does not analyze model transition structure may not be able to compute any result. However, this feature of not analyzing the model allows the application of statistical verification methods (including ours) for the purpose of estimating the probability of properties in systems for which pre-analysis of system-model is not possible due to prohibitively large state-space; or for systems for which only sample simulations of the system can be generated (as needed).

Consider the DTMC model in Figure 6.2. The start state of the model is s_0 and each state is annotated with the property (φ_1, φ_2) that holds at that state. As in the illustrative example introduced in Section 1.1, the dotted segment, denoted by DS, represents some complicated transition structure on n different states (an exact description of the model is available at Jennings et al. (2010)). All states in DS satisfy φ_1. The objective is to compute $\mathsf{P}(s_0, \varphi_1 \; \mathtt{U} \; \varphi_2)$. There are three paths $s_0, s_3, s_3, \ldots$, $s_0, s_4, s_4, \ldots$ and $s_0, t_0, \ldots, t_m, s_4, \ldots$ that satisfy $\varphi_1 \; \mathtt{U} \; \varphi_2$; therefore, $\mathsf{P}(s, \varphi_1 \; \mathtt{U} \; \varphi_2) = 0.2 + 0.2 + 0.4 \times 0.65 = 0.66$. Observe that there exists an infinite path in the model with positive probability where every state satisfies $(\varphi_1 \wedge \neg\varphi_2)$. In fact, $\lim_{k\to\infty} \mathsf{P}(s_0, \psi_k) = 0.86$. As PRISM's statistical method requires the above limit to be 1, it

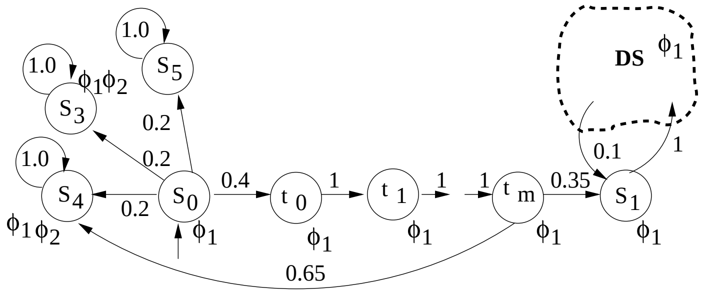


Figure 6.2 A modification of the illustrative example (Figure 1.1).

fails to compute the estimate of $\mathsf{P}(s_0, \varphi_1 \texttt{ U } \varphi_2)$. Similarly, our proposed method `U2B` will fail to terminate when $\epsilon_0 < 0.14$. More precisely, *Phase I* of the `U2B` method will fail to terminate. In the following, we present a strategy and a corresponding heuristic to address this problem of non-termination.

6.1 Rationale

From the semantics of `PCTL` (Section 2.2), the probability of satisfying $\varphi_1 \texttt{ U } \varphi_2$ can be expressed in terms of satisfying $\varphi_1 \texttt{ U}^{\leq k} \varphi_2$ as follows:

$$\lim_{k\to\infty} \mathsf{P}(s, \varphi_1 \texttt{ U}^{\leq k} \varphi_2) = \mathsf{P}(s, \varphi_1 \texttt{ U } \varphi_2), \text{ i.e., } \mathsf{P}(s, \varphi_1 \texttt{ U}^{\leq \infty} \varphi_2) = \mathsf{P}(s, \varphi_1 \texttt{ U } \varphi_2) \tag{6.2}$$

Next, suppose there exists a method by which a k_* can be computed such that

$$\forall \Delta_k \geq 1 : \widehat{\mathsf{P}}(s, \varphi_1 \texttt{ U}^{\leq k_*} \varphi_2) = \widehat{\mathsf{P}}(s, \varphi_1 \texttt{ U}^{\leq k_* + \Delta_k} \varphi_2) \tag{6.3}$$

We refer to any k_* that satisfies the above equation as an *unbounded-plateau-detector* (the valuation of estimate $\widehat{\mathsf{P}}(s, \varphi_1 \texttt{ U}^{\leq k} \varphi_2)$ remains unchanged for all $k \geq k_*$). Note that there is at most one unbounded plateau and an infinite number of unbounded-plateau-detectors. The above equation implies that $\widehat{\mathsf{P}}(s, \varphi_1 \texttt{ U}^{\leq k_*} \varphi_2) = \widehat{\mathsf{P}}(s, \varphi_1 \texttt{ U}^{\leq \infty} \varphi_2)$. Proceeding further, we prove the following theorem which forms the basis of our method.

Theorem 2. *Given any precision parameter $\epsilon_1 > 0$ and confidence parameter $\delta_1 > 0$,*

$$Pr\left(\mid \widehat{P}(s, \varphi_1 \ U^{\leq k_*} \ \varphi_2) - P(s, \varphi_1 \ U \ \varphi_2) \mid > \ \epsilon_1\right) \ \leq \ \delta_1$$

where k_* is an unbounded-plateau-detector and $\delta_1 \geq 2e^{-2N_1(\epsilon_1)^2}$.

Proof.

$$\begin{aligned}
|\,\widehat{\mathsf{P}}(s,\varphi_1\ \mathtt{U}^{\leq k_*}\ \varphi_2) - \mathsf{P}(s,\varphi_1\ \mathtt{U}\ \varphi_2)\,| &\leq |\,\widehat{\mathsf{P}}(s,\varphi_1\ \mathtt{U}^{\leq k_*}\ \varphi_2) - \mathsf{P}(s,\varphi_1\ \mathtt{U}^{\leq\infty}\ \varphi_2)\,| \\
&\quad + |\,\mathsf{P}(s,\varphi_1\ \mathtt{U}^{\leq\infty}\ \varphi_2) - \mathsf{P}(s,\varphi_1\ \mathtt{U}\ \varphi_2)\,| \\
&\leq |\,\widehat{\mathsf{P}}(s,\varphi_1\ \mathtt{U}^{\leq k_*}\ \varphi_2) - \mathsf{P}(s,\varphi_1\ \mathtt{U}^{\leq\infty}\ \varphi_2)\,| \\
&\qquad \text{using Equation 6.2}
\end{aligned}$$

Therefore,

$$\begin{aligned}
|\,\widehat{\mathsf{P}}(s,\varphi_1\ \mathtt{U}^{\leq k_*}\ \varphi_2) - \mathsf{P}(s,\varphi_1\ \mathtt{U}\ \varphi_2)\,| &\leq |\,\widehat{\mathsf{P}}(s,\varphi_1\ \mathtt{U}^{\leq k_*}\ \varphi_2) - \widehat{\mathsf{P}}(s,\varphi_1\ \mathtt{U}^{\leq\infty}\ \varphi_2)\,| \\
&\quad + |\,\widehat{\mathsf{P}}(s,\varphi_1\ \mathtt{U}^{\leq\infty}\ \varphi_2) - \mathsf{P}(s,\varphi_1\ \mathtt{U}^{\leq\infty}\ \varphi_2)\,| \\
&\leq |\,\widehat{\mathsf{P}}(s,\varphi_1\ \mathtt{U}^{\leq\infty}\ \varphi_2) - \mathsf{P}(s,\varphi_1\ \mathtt{U}^{\leq\infty}\ \varphi_2)\,| \\
&\qquad \text{using Equation 6.3}
\end{aligned} \tag{6.4}$$

From Lemma 1, the following holds for any k, $N_1 \geq 1$ and $\epsilon_1 > 0$

$$Pr\left(\sup_{k\geq 1} |\,\widehat{\mathsf{P}}(s,\varphi_1\ \mathtt{U}^{\leq k}\ \varphi_2) - \mathsf{P}(s,\varphi_1\ \mathtt{U}^{\leq k}\ \varphi_2)\,| > \epsilon_1\right) \leq 2e^{-2N_1(\epsilon_1)^2}$$

where $\widehat{\mathsf{P}}(s,\varphi_1\ \mathtt{U}^{\leq k}\ \varphi_2)$ is the proportion of paths (starting from s) in N_1 samples that satisfy $\varphi_1\ \mathtt{U}^{\leq k}\ \varphi_2$. Therefore, from Equation 6.4

$$\begin{aligned}
&Pr\left(|\,\widehat{\mathsf{P}}(s,\varphi_1\ \mathtt{U}^{\leq\infty}\ \varphi_2) - \mathsf{P}(s,\varphi_1\ \mathtt{U}^{\leq\infty}\ \varphi_2)\,| > \epsilon_1\right) \leq \delta_1, \text{ i.e.,} \\
&Pr\left(|\,\widehat{\mathsf{P}}(s,\varphi_1\ \mathtt{U}^{\leq k_*}\ \varphi_2) - \mathsf{P}(s,\varphi_1\ \mathtt{U}\ \varphi_2)\,| > \epsilon_1\right) \leq \delta_1
\end{aligned}$$

where $\delta_1 \geq 2e^{-2N_1(\epsilon_1)^2}$.

It follows that if $N_1 \geq \frac{1}{2\epsilon_1^2}log(\frac{\delta_1}{2})$ samples are considered to compute a k_* such that Equation 6.3 is satisfied, then the proportion of paths that satisfy $\varphi_1\ \mathtt{U}^{\leq k_*}\ \varphi_2$ in N_1 samples (i.e., $\widehat{\mathsf{P}}(s,\varphi_1\ \mathtt{U}^{\leq k_*}\ \varphi_2)$) estimates $\mathsf{P}(s,\varphi_1\ \mathtt{U}\ \varphi_2)$ with precision parameter ϵ_1 and confidence parameter δ_1. □

The above theorem provides a sound roadmap to estimate $\mathsf{P}(s,\varphi_1\ \mathtt{U}\ \varphi_2)$ and it does not require that $\lim_{k\to\infty}\mathsf{P}(s,\psi_k) \geq 1-\epsilon_0$ (as is necessary for the termination of *Phase I* of our `U2B` method). However, this roadmap suffers from the drawback that it assumes the existence of a method for computing k_*, an unbounded-plateau-detector. No such method for computing k_* can be realized in practice. As such, we propose a heuristic for estimating k_* which ensures termination of `U2B` at the cost of precision (i.e., the resulting $\widehat{\mathsf{P}}(s, \varphi_1 \ \mathtt{U}\ \varphi_2)$ may not be an estimate within pre-specified error bound and confidence parameter) when $\lim_{k\to\infty} \mathsf{P}(s, \psi_k) \not\gtrsim 1 - \epsilon_0$.

6.2 U2B_P: Heuristic for Computing Plateau-Detector using the D-Graph

If in a D-Graph the valuation of $F(k) = \mathsf{P}(s, \psi_k)$ for a range of values of k (see Figure 6.1) remains unaltered, then a plateau in the corresponding valuation of $\mathsf{P}(s, \varphi_1 \ \mathtt{U}^{\leq k}\ \varphi_2)$ occurs for the same k values. Recall that *Phase I* of `U2B` does not terminate when there exists an unbounded plateau in $F(\cdot)$ such that the value of the plateau is less than $1 - \epsilon_0$. For instance $F(\cdot)$ for the model in Figure 6.2 has an unbounded plateau with valuation 0.86.

The basis of the heuristic, therefore, is as follows. An estimate $\hat{k}_*$ of k_* is said to be "good" if a "long" plateau in $\hat{F}_{N_1}(k) = \widehat{\mathsf{P}}(s, \psi_k)$ (estimate of $F(k)$ from N_1 samples) is detected starting from $k = \hat{k}_*$. In other words, given Γ, a pre-specified length of a plateau, $\hat{k}_*$ is an estimate of k_* if $\forall k : \hat{k}_* \leq k \leq \hat{k}_* + \Gamma$, the valuation of $\hat{F}_{N_1}(k)$ remains unaltered.

$$\forall k : k_* \leq k < (k_* + \Gamma) : \hat{F}_{N_1}(k) = \hat{F}_{N_1}(k+1) \Rightarrow \widehat{\mathsf{P}}(s, \varphi_1 \ \mathtt{U}^{\leq k}\ \varphi_2) = \widehat{\mathsf{P}}(s, \varphi_1 \ \mathtt{U}^{\leq k+1}\ \varphi_2) \quad (6.5)$$

The precision of this method of estimation depends on the length, Γ, of the detected plateau; a longer plateau (larger Γ) is likely to provide a better estimate $\hat{k}_*$. For example, in Figure 6.1, if Γ is greater than $k_2 - k_1$, $k_3 - k_4$, and $k_6 - k_5$, then $\hat{k}_*$ will be indeed an unbounded-plateau-detector; otherwise, one of the local plateaus may result in an incorrect $\hat{k}_*$ computation. In the following, we provide a heuristic to obtain a suitable valuation for Γ based on the valuations of k.

Let k_i be the current valuation of the simulation length and k_j be the smallest value of k for which $\hat{F}_{N_1}(k_i) = \hat{F}_{N_1}(k_j)$. The objective is decide whether the difference $k_i - k_j$ is a long enough ($\geq \Gamma$) to infer that a *global* plateau is detected. Assuming that $F(\cdot)$ increases at a constant rate given by $F(k_j)/k_j$, we estimate the k_x for which $F(k_x) = 1$

$$\frac{1 - F(k_j)}{k_x - k_j} = \frac{1}{k_x} \quad \Leftrightarrow \quad k_x - k_j = k_j\left(\frac{1}{F(k_j)} - 1\right)$$

We say that $\Gamma = (k_j + \texttt{AF})\gamma^{\texttt{simpoly}}$ where

- $\texttt{AF}$ is the additive factor $(= 10)$.
- $\gamma = \texttt{MINMAX}\left(\texttt{LB}, \frac{1}{\hat{F}_{N_1}(k_j)} - 1, \texttt{UB}\right)$,
- `simpoly` is a user-specified parameter (default value is 1),
- $\texttt{LB} = 2$ and $\texttt{UB} = 5$ are the pre-specified lower- and upper-bounds of γ, and
- $\hat{F}_{N_1}(\cdot)$ is the estimator of $F(\cdot)$ (see Section 4.3.1)

The value k_j is said to be an estimate $\hat{k}_*$ when $k_i - k_j > \Gamma$ and $\hat{F}_{N_1}(k_i) = \hat{F}_{N_1}(k_j)$. Note that we have arbitrarily set the three parameters `AF`, `UB` and `LB`. `AF` is used to ensure that a long enough plateau is considered even when k_j is small (e.g., $k_j = 1$). `LB` (and `UB`) ensures that the required plateau length grows reasonably by exponents of 2 (or 5) if the slope $\hat{F}_{N_1}(k_j)/k_j$ is too large (or too small). We have allowed users the control the length of the plateau using the exponent `simpoly`.

Consider an example illustration in Figure 6.1. At k_1, the value of Γ is $k_4 - k_1$ (assume that there is no pre-set `LB`, `UB`, `AF`), i.e., if the value of $\hat{F}_{N_1}(\cdot)$ remains unaltered between k_1 and k_4 (inclusive), we say that a plateau is identified starting from k_1 and the corresponding value of $\widehat{\mathsf{P}}(s, \varphi_1\ \mathtt{U}^{\leq k_1}\ \varphi_2)$ is identified as the $\widehat{\mathsf{P}}(s, \varphi_1\ \mathtt{U}\ \varphi_2)$ (see Equation 6.5). Similarly at k_3, Γ computed using the above formula is $k_7 - k_3$.

We have developed a method, referred to as `U2B_P`, based on the above heuristic to compute $k_j = \hat{k}_*$. In `U2B_P`, if a plateau in $\hat{F}_{N_1}(\cdot)$ is detected starting from k_j and if $\hat{F}_{N_1}(k_j) \not> 1 - \epsilon_0$, then $\widehat{\mathsf{P}}(s, \varphi_1\ \mathtt{U}^{\leq k_j}\ \varphi_2)$ is returned as the estimate of $\mathsf{P}(s, \varphi_1\ \mathtt{U}\ \varphi_2)$ (see Theorem 2). $\widehat{\mathsf{P}}(s, \varphi_1\ \mathtt{U}^{\leq k_j}\ \varphi_2)$ is the proportion of paths in N_1 samples that satisfy $\varphi_1\ \mathtt{U}^{\leq k_j}\ \varphi_2$. On the other hand, if $\hat{F}_{N_1}(k_j) > 1 - \epsilon_0$, then `U2B_P` is said to have successfully estimated k_0 $(= k_j)$ and therefore `U2B_P` reduces to `U2B` and invokes *Phase II* computation (as described in `U2B` method; see Section 4.3.2).

Algorithm 3 Implementation of `U2B_P`

```
1: procedure PLATEAUKSTAR($N_1, \epsilon_o, \psi, simpoly$)
                         ▷ $N_1$: Total number of samples for phase 1 (see U2B Step 1(a))
                         ▷ $\epsilon_0$: Error bound for phase 1 (see U2B Step 1(c))
                         ▷ $\psi := (\varphi_1$ U $\varphi_2) \vee (\neg\varphi_2$ U $\neg\varphi_1 \wedge \neg\varphi_2)$
                         ▷ simpoly: Exponent for plateau length calculation
2:     $k := 0$;                                        ▷ Initial length of paths
3:     lastk := 0;                                      ▷ last value of $k$ for which $\psi_k$ was satisfied
4:     workingSet := add $N_1$ copies of start state;
5:     trueCount := 0;                                  ▷ Initial number of paths that satisfy $\psi_k$
6:     satCount := 0;                                   ▷ Initial number of paths that satisfy $\varphi_1$ U$^{\leq k}$ $\varphi_2$
7:     while (1) do
8:         for each state $\in$ workingSet that satisfies $\neg\varphi_1 \vee \varphi_2$ do    ▷ i.e., path satisfies $\psi_k$
9:             remove state from workingSet;
10:            trueCount++;
11:            if (the state satisfies $\varphi_2$) then
12:                $\boxed{\texttt{satCount++;}}$                 ▷ update satCount
13:            end if
14:            $\boxed{\texttt{lastk} := k;}$                      ▷ record value of $k$
15:        end for
16:        if ($N_1(1 - \epsilon_0) \leq$ trueCount) then
17:            return $k$;         ▷ trueCount $\geq N_1(1-\epsilon_0)$, i.e., $\widehat{\mathsf{P}}(s,\psi_k) = \frac{\texttt{truecount}}{N_1} \geq 1-\epsilon_0$
                                   ▷ Invoke Phase II of U2B using $k$
18:        end if
19:        $k := k$++;
20:        replace current states in workingSet with randomly obtained next states;
21:        if (trueCount $> 0$) then                   ▷ Calculation of plateau length
22:            $\boxed{\Gamma := (\texttt{lastk + 10}) \times [\texttt{MINMAX}(2, N_1/\texttt{trueCount} - 1, 5)]^{\texttt{simpoly}};}$
23:        end if
24:        if ($\Gamma > 0$) && ($k >$ lastk + $\Gamma$) then    ▷ Condition for plateau: heuristic
25:            return $k$ and satCount/$N_1$ as $\widehat{\mathsf{P}}(s, \varphi_1$ U $\varphi_2)$
26:        end if
27:    end while
28: end procedure
```

Remark 3. *As `U2B_P` is based on a heuristic, it cannot provide any correctness guarantees. We give users the option to deploy either `U2B` (with the Algorithm 2 implementation for* Phase I*) or `U2B_P`. In the event that `U2B` fails to return an estimated probability within the amount of time the user is willing to wait, the user can terminate the process and deploy `U2B_P`. We have not deployed `U2B_P` in any of the experimental results presented in Chapter 8.*

The `U2B_P` method is realized as follows in Algorithm 3. In addition to keeping track of the proportion of paths that satisfy ψ_k (`trueCount`, i.e., $\hat{F}_{N_1}(k)$), the algorithm also keeps information regarding the number of paths that satisfy φ_1 U$^{\leq k}$ φ_2 (`satCount` in Lines 6, 12).

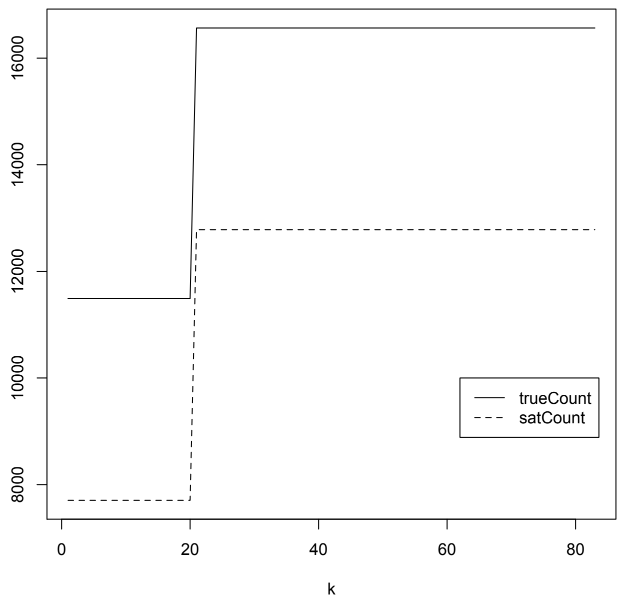


Figure 6.3 Execution of Algorithm 3 for `DTMC` model in Figure 6.2: `trueCount` and `satCount` vs. k.

The value of k for which there is some change in the valuation of `trueCount` is recorded in `lastk` (Line 14). Γ is computed at Line 22 following the above description. If `trueCount` does not change for a large range of values of k (i.e, the difference between current value of k and `lastk` is $> \Gamma$), then for the same range the value of `satCount` will also not change (Equation 6.5). If a plateau is detected in the value of `trueCount` (i.e., $F_{N_1}(\cdot)$), then `satCount`$/N_1$ is returned as an estimate for $\mathsf{P}(s, \varphi_1 \texttt{ U } \varphi_2)$ along with a warning to the user that a heuristic has been used to satisfy Equation 6.3 (Line 25). The computation can be re-done with larger values of Γ, if the user so chooses, by increasing `simpoly`. If the termination condition of the while-loop at Line 8 is satisfied, then it implies that $\hat{F}_{N_1}(k)$ is ϵ_0 close to 1. In that case, the algorithm returns k as k_0 (Line 17) and *Phase II* of `U2B` is invoked.

6.2.1 Discussion: `U2B` vs `U2B_P`

Consider the example model in Figure 6.2. The `U2B` method will fail to terminate and estimate $\mathsf{P}(s, \varphi_1 \ \mathtt{U}\ \varphi_2)$ for this model because the limit[1] $\lim_{k\to\infty} \mathsf{P}(s, \psi_k) = 0.86 < 1 - \epsilon_0$ where $\epsilon_0 = 0.0025$. That is, $\mathsf{P}(s, \psi_k)$ or $F(k)$ has an unbounded plateau for some value of k and the valuation of $\mathsf{P}(s, \psi_k)$ at the plateau is $< 1 - \epsilon_0$.

The `U2B_P` method as implemented in Algorithm 3, on the other hand, terminates and successfully estimates $\mathsf{P}(s_0, \varphi_1 \ \mathtt{U}\ \varphi_2)$. In the example, we have considered $m = 20$ (number of t_i-states in the model) and used `simpoly` $= 1$. Algorithm 3 identifies two plateaus–one where the simulation paths end up in the states $t_0, \ldots, t_m$ (local plateau) and the other where the simulation paths end up in the `DS` (global plateau). Figure 6.3 shows these plateaus in terms of the changes in the valuation of the `trueCount` (number of paths with decided result), `satCount` (number of paths that satisfy $\varphi_1 \ \mathtt{U}\ \varphi_2$).

The execution starts with $k = 0$ and $|\mathtt{workingSet}| = N_1 = 19,171$ ($N_1 = 19,171$ for $\epsilon_0 = 0.0025$ and $\delta = 0.01$). The value of `trueCount` remains at $11,492$ until the value of k becomes 20, i.e., $\forall k \in [1, 20] : \hat{F}_{N_1}(k) = \widehat{\mathsf{P}}(s, \psi_k) = 0.59942$. This is because the length of paths considered is ≤ 20 and as such the transitions t_m to s_4 and t_m to s_1 are yet to be explored. However, the length (20) of this plateau in $\hat{F}_{N_1}(\cdot)$ is not considered to be sufficient as per the heuristic because $\forall \mathtt{lastk} \in [1, 19] : \mathtt{lastk} + \Gamma \geq 23$ (see the computation of Γ described in Section 6.2). That is, if $\hat{F}_{N_1}(\cdot)$ remained unaltered for paths of length 23, then `U2B_P` would infer that a reasonably large size plateau have been detected and, therefore, would terminate. In the current example, that does not happen as $\hat{F}_{N_1}(20) < \hat{F}_{N_1}(21)$[2].

The value of `trueCount` remains at $16,566$ for $k \geq 21$, i.e., the last time `trueCount` value changes is when $k = 21$. In this case, when `lastk` $= 21$, $\Gamma = 62$. In other words, if the `trueCount` value does not change for k between 21 and $21 + 62 = 83$, the `U2B_P` terminates. Observe that changes in `satCount` follow a similar pattern as `trueCount`; a

[1] Recall that ψ_k denotes the property stating that $\varphi_1 \ \mathtt{U}\ \varphi_2$ is decided in k steps. The $\lim_{k\to\infty} \mathsf{P}(s, \psi_k) \geq 1 - \epsilon_0$ (see Equation 6.1) is required for the termination of Algorithm 2.

[2] If m is increased to > 23 and `simpoly` is set to 1, then the heuristic based method `U2B_P` would terminate, incorrectly infer a plateau, and estimate $\mathsf{P}(s_0, \varphi_1 \ \mathtt{U}\ \varphi_2)$ incorrectly. That is why `U2B_P` produces a warning message on termination and recommends that the user re-run `U2B_P` using different values of `simpoly`.

plateau in the valuation of `trueCount` (i.e., $\hat{F}_{N_1}(k)$) implies a plateau in the valuation of `satCount` (i.e., $\widehat{\mathsf{P}}(s_0, \varphi_1 \mathtt{U}^{\leq k} \varphi_2)$). As such, on termination `U2B_P` returns $\widehat{\mathsf{P}}(s_0, \varphi_1 \mathtt{U}^{\leq \mathtt{lastk}} \varphi_2) =$ `satCount`$/N_1 = 0.66665$ as the estimate for $\mathsf{P}(s_0, \varphi_1 \mathtt{U} \varphi_2)$; recall that $\mathsf{P}(s, \varphi_1 \mathtt{U} \varphi_2) = 0.66$.

CHAPTER 7 Application to Continuous Time Markov Chain Models

So far, we have shown the application of U2B in the context of probabilistic model checking of unbounded until properties against DTMC models. In this chapter, we discuss its applicability in verifying similar properties for CTMC models. Unlike a DTMC model where transition from one state to another captures the probability of a *discrete* time step, a Continuous Time Markov Chain (CTMC) model describes the probability with which a system evolves from one state (configuration) to another within t time units. Formally, a CTMC is defined as:

Definition 2. *A Continuous Time Markov Chain CTMC* $= (S, s_I, R, L)$*, where* S *is a finite set of states,* $s_I \in S$ *is the initial or start state,* $R : S \times S \to \mathbb{R}_{\geq 0}$ *is the rate matrix, and* $L : S \to \mathcal{P}(AP)$ *is the labeling function which labels each state with a set of atomic propositions* $\subseteq AP$ *that holds in that state.*

The rate matrix R in the above definition denotes the rate at which the system evolves from one state to another, while for any $s \in S$, $E(s) = \Sigma_{s' \in S} R(s, s')$ denotes the rate at which the system moves out of the state s. This is used as the parameter to the exponential distribution capturing the probability $1 - e^{-E(s).t}$ of taking an outgoing transition from s within t time units. If $R(s, s') > 0$ for multiple s', then there exists a *race* between the moves from s to its multiple next states. The probability $T(s, s')$ with which a state s in CTMC moves to a state s' in single step is equal to the probability that the delay of moving from s to s' is less than those for any other moves from s; $T(s, s') = R(s, s')/E(s)$. Based on this observation, a CTMC *contains* an embedded DTMC which captures the probability of each transition independent of the time at which it occurs.

Definition 3. *Baier et al. (2003) Given a CTMC* $= (S, s_I, R, L)$*, the corresponding embedded*

DTMC, emb(DTMC) $= (S, s_I, T, L)$, *where* $T : S \times S \to [0,1]$ *such that*

$$T(s,s') = \begin{cases} R(s,s')/E(s) & \textit{if } E(s) > 0 \\ 1 & \textit{if } E(s) = 0 \textit{ and } s' = s \\ 0 & \textit{if } E(s) = 0 \textit{ and } s' \neq s \end{cases}$$

Paths and Probability Measures. A path σ in CTMC is a finite or infinite sequence of tuples $(s_0t_0, s_1t_1, s_2t_2, \ldots)$ where t_i denotes the amount of time the system remains in state s_i. As in DTMC, the set of all infinite paths starting from state s is denoted by $Path_{\texttt{CTMC}}(s)$. Let $\sigma[i]$ denote the i-th state in the path, σ^i denote the suffix of σ starting from $\sigma[i]$ and $\sigma@t$ denote the state at time t. Then, $\sigma@t = \sigma[i]$ where $i = \texttt{MIN}(k \mid t \leq \sum_{j=0}^{k} t_j)$. We will use $\sigma_{\mathsf{emb(DTMC)}}$ to denote $(s_0, s_1, s_2, \ldots)$ which represents a path corresponding to σ in the emb(DTMC). Observe that $\sigma[i] = \sigma_{\mathsf{emb(DTMC)}}[i]$ for $i \geq 0$.

$C(s_0I_0, s_1I_1, \ldots, s_{n-1}I_{n-1}, s_n)$ denotes the cylinder set consisting of all paths of the form $\sigma = (s_0t_0, s_1t_1, \ldots) \in Path_{\texttt{CTMC}}(s_0)$ such that $\sigma[i] = s_i$ and $t_i \in I_i$ for $i < n$ and $I \in \mathbb{R}_{\geq 0}$. Proceeding further, $Pr_{\texttt{CTMC}}(C(s_0I_0, s_1I_1, \ldots, s_n))$ is recursively defined as follows.

$$Pr_{\texttt{CTMC}}(C(s_0I_0, s_1I_1, \ldots, s_n)) \begin{cases} 1 \text{ if } n = 0 \\ Pr_{\texttt{CTMC}}(C(s_0I_0, s_1I_1, \ldots, s_{n-1}).T(s_{n-1}, s_n). \\ \qquad \displaystyle\int_{I_{n-1}} E(s_{n-1}).e^{-E(s_{n-1})t} dt \\ \qquad\qquad \text{otherwise} \end{cases} \tag{7.1}$$

Observe that $T(s_{n-1}, s_n)$ in the above equation corresponds to probability of moving from s_{n-1} to s_n in the emb(DTMC) (Definition 3). The last term in the above equation captures the probability of exiting s_{n-1} in the interval I_{n-1}.

CTMC properties. Properties of interest for CTMC include transient and steady state properties which are expressed in Continuous Stochastic Logic (CSL), developed by Aziz et al. (1996). Here, we focus on verification of a specific type of CSL property: the time unbounded until property of the form $\mathtt{P}_{\bowtie r}(\varphi_1 \ \mathtt{U}^{[0,\infty)} \ \varphi_2)$ which is satisfied by $s \in S$ such that

$\mathsf{P}_{\mathtt{CTMC}}(s, \varphi_1 \ \mathtt{U}^{[0,\infty)} \ \varphi_2) \bowtie r$, where

$$\begin{aligned} &\mathsf{P}_{\mathtt{CTMC}}(s, \varphi_1 \ \mathtt{U}^{[0,\infty)} \ \varphi_2) = Pr_{\mathtt{CTMC}}(\{\sigma \in Path_{\mathtt{CTMC}}(s) : \sigma \models \varphi_1 \ \mathtt{U}^{[0,\infty)} \ \varphi_2\}), \text{ and} \\ &\sigma \models \varphi_1 \ \mathtt{U}^{[0,\infty)} \ \varphi_2 \Leftrightarrow \exists a \in [0,\infty) : \sigma@a \models \varphi_2 \ \wedge \ \forall b < a : \sigma@b \models \varphi_1) \end{aligned} \tag{7.2}$$

We will use the fact that $\mathsf{P}_{\mathtt{CTMC}}(s, \varphi_1 \ \mathtt{U}^{[0,\infty)} \ \varphi_2)$ is equal to $\mathsf{P}(s, \varphi_1 \ \mathtt{U} \ \varphi_2)$ in the corresponding $\mathtt{emb}(\mathtt{DTMC})$. For the completeness of discussion, we present here the proof of the above statement.

$$\begin{aligned} \mathsf{P}_{\mathtt{CTMC}}(s, \varphi_1 \ \mathtt{U}^{[0,\infty)} \ \varphi_2) &= Pr_{\mathtt{CTMC}}(\{\sigma \in Path_{\mathtt{CTMC}}(s) : \sigma \models \varphi_1 \ \mathtt{U}^{[0,\infty)} \ \varphi_2\}) \\ &= \sum_{\sigma \in \Upsilon} Pr_{\mathtt{CTMC}}(\sigma) \\ &\quad \text{where } \Upsilon = \{\sigma \in Path_{\mathtt{CTMC}}(s) : \sigma \models \varphi_1 \ \mathtt{U}^{[0,\infty)} \ \varphi_2\} \end{aligned}$$

In the above, consider any path $\sigma = s_0 t_0, s_1 t_1, \ldots$. From Equation 7.2, $\exists a \in [0,\infty) : \sigma@a \models \varphi_2$ and $\forall b < a : \sigma@b \models \varphi_1$. Let $\sigma@a = \sigma[k] = s_k$. Therefore,

$$Pr_{\mathtt{CTMC}}(\sigma^k) = 1 \text{ as } \sigma@a = \sigma[k] \ \wedge \ \sigma@a \models \varphi_2 \tag{7.3}$$

and $\forall i : 0 \leq i < k$

$$Pr_{\mathtt{CTMC}}(\sigma^i) \ = \ \int_0^\infty T(s_i, s_{i+1}).E(s_i).e^{-E(s_i).t_i}.Pr_{\mathtt{CTMC}}(\sigma^{i+1}) \ dt_i \tag{7.4}$$

From the above,

$$Pr_{\mathtt{CTMC}}(\sigma^0) \ = \ \int_0^\infty T(s_0, s_1).E(s_0).e^{-E(s_0).t_0}.Pr_{\mathtt{CTMC}}(\sigma^1) \ dt_0 \ = \ T(s_0, s_1).Pr_{\mathtt{CTMC}}(\sigma^1)$$

and

$$Pr_{\mathtt{CTMC}}(\sigma^1) \ = \ \int_0^\infty T(s_1, s_2).E(s_1).e^{-E(s_1).t_1}.Pr_{\mathtt{CTMC}}(\sigma^2) \ dt_1 \ = \ T(s_1, s_2).Pr_{\mathtt{CTMC}}(\sigma^2)$$

Proceeding further,

$$Pr_{\mathtt{CTMC}}(\sigma) \ = Pr_{\mathtt{CTMC}}(\sigma^0) = T(s_0, s_1).T(s_1, s_2).\ldots.T(s_{k-1}, s_k) \ = \ \prod_{i=0}^{k-1} T(s_i, s_{i+1})$$

Note that for any σ in CTMC there is a corresponding path $\sigma_{\mathsf{emb(DTMC)}}$ in the emb(DTMC) such that $\sigma_{\mathsf{emb(DTMC)}}[i] = \sigma[i]$ for all $i \geq 0$, and $\sigma \models \varphi_1 \ \mathtt{U}^{[0,\infty)} \ \varphi_2 \Rightarrow \sigma_{\mathsf{emb(DTMC)}} \models \varphi_1 \ \mathtt{U} \ \varphi_2$ in the emb(DTMC). Hence,

$$Pr_{\mathtt{CTMC}}(\sigma) \ = \ \prod_{i=0}^{k-1} T(s_i, s_{i+1}) \ = \ Pr(\sigma_{\mathsf{emb(DTMC)}}) \text{ (see Equation 2.1 in Section 2.1)} \tag{7.5}$$

Furthermore,

$$\begin{aligned} &\sigma \in Path_{\mathtt{CTMC}}(s) \Rightarrow \sigma_{\mathsf{emb(DTMC)}} \in Path_{\mathsf{emb(DTMC)}}(s) \text{ and} \\ &\pi \in Path_{\mathsf{emb(DTMC)}}(s) \Rightarrow \exists \sigma \in Path(s) : \sigma_{\mathsf{emb(DTMC)}} = \pi \end{aligned} \tag{7.6}$$

where $Path_{\mathsf{emb(DTMC)}}(s)$ denotes the set of paths starting from s in the emb(DTMC). Recall that, $\Upsilon = \{\sigma \in Path_{\mathtt{CTMC}}(s) : \sigma \models \varphi_1 \ \mathtt{U}^{[0,\infty)} \ \varphi_2\}$. Let, $\Xi = \{\sigma_{\mathsf{emb(DTMC)}} : \sigma \in \Upsilon\}$. From the above, it follows that $\Xi = \{\pi : \pi \in Path_{\mathsf{emb(DTMC)}}(s) \ \wedge \ \pi \models \varphi_1 \ \mathtt{U} \ \varphi_2\}$ Therefore, from Equations 7.5 and 7.6

$$\begin{aligned} \mathsf{P}_{\mathtt{CTMC}}(s, \varphi_1 \ \mathtt{U}^{[0,\infty)} \ \varphi_2) \ &= \ \sum_{\sigma \in \Upsilon} Pr_{\mathtt{CTMC}}(\sigma) \ = \ \sum_{\pi \in \Xi} Pr(\pi) \\ &= \ \mathsf{P}(s, \varphi_1 \ \mathtt{U} \ \varphi_2) \text{ in the emb(DTMC) (see Section 2.1)} \end{aligned}$$

Hence, a CSL property of the form $\mathtt{P}_{\bowtie r}(\varphi_1 \ \mathtt{U}^{[0,\infty)} \ \varphi_2)$ can be verified at any state s in a CTMC by verifying satisfiability of $\mathtt{P}_{\bowtie r}(\varphi_1 \ \mathtt{U} \ \varphi_2)$ in the corresponding embedded DTMC, emb(DTMC), at the same state s. That is, our U2B algorithm is applicable in the verification of $\mathtt{P}_{\bowtie r}(\varphi_1 \ \mathtt{U}^{[0,\infty)} \ \varphi_2)$ for CTMC models.

CHAPTER 8 Overview of PRISM-U2B Tool

We have realized the U2B and U2B_P methods in a tool called PRISM-U2B, which is developed leveraging the existing implementation of the PRISM model checker, as described by Hinton et al. (2006) (available under GNU GPL). The work described here is implemented based on PRISM version 3.3.1. Recently, pre-release versions of PRISM 4 have become available. Implementation in PRISM 4 is discussed in 10.2. Figure 8.1 presents an overview of the various modules used in PRISM-U2B. Observe that PRISM-U2B re-uses PRISM's user interface, parsers, and simulation engine. Specifically, PRISM-U2B uses PRISM's input specification language and presents a user interface similar to PRISM. This significantly minimizes the cognitive burden to learn and understand the usage of PRISM-U2B. While simple changes to the PRISM user interface were sufficient to allow for the additional error bounds and confidence parameters used in the two phases of PRISM-U2B, the internal changes to the PRISM simulator are significant. In the following, we discuss the distinguishing aspects of PRISM-U2B with respect to realization of sample path generation.

8.1 Simulator overview

The core of PRISM's statistical method is the simulation generator engine. It takes as input (a) the model (e.g., DTMC) given in terms of rules and probabilities for transitions as translated from PRISM's model specification language, and (b) the property (e.g., in PCTL) translated from PRISM's property specification language. Notably, PRISM does not generate a concrete representation of the model in memory; rather, it reads the model specification as rules that can later be translated into transition probabilities. The simulator then calculates the number of paths (N) required to compute the probability estimate given the user-specified error bound

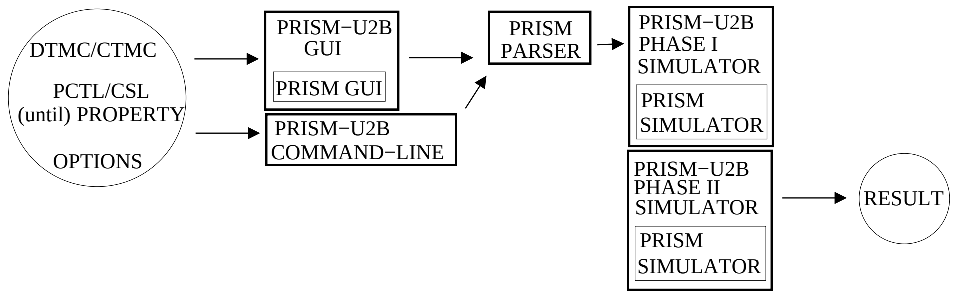


Figure 8.1 Architectural overview and module dependencies in `PRISM-U2B`.

and confidence parameter. Finally, it iterates N times, in each iteration randomly generating one path up to a pre-specified maximum length k (by default, $10,000$) and returns as the probability estimate the proportion among these N paths that satisfy the given property. In the event that the simulator obtains a path in which the given property is not decided (i.e., satisfiability and un-satisfiability cannot be inferred), `PRISM` fails and requests that the user specify a larger value for k.

Note that the methods described here require selecting the next state randomly based on the rules describing the model. This is generally accomplished by generating a uniform random variable and selecting the next state based on the probability distribution from the model. `PRISM` version 3 uses the **`rand`** and **`srand`** functions from the C standard library to generate these values. Younes (2005a) gives more commentary on the application of various random number generators to probabilistic model checking, and recommends the Mersenne twister, developed by Matsumoto and Nishimura (1998). `PRISM` version 4 uses the Mersenne twister.

The `PRISM` simulator currently does not support models with multiple initial states; `PRISM-U2B` inherits this limitation.

8.2 Implementation of `PRISM-U2B`

As described in Chapter 4, `U2B` performs model checking in two phases. The changes required to implement `U2B` are described below. When calls to the simulator are made, the `U2B` implementation checks for unbounded until properties. If no such properties are found,

Algorithm 4 PRISM's sample path generation

```
1: procedure $(M, s_0, \psi, maxLength)$
                                        ▷ $M$: Model transition rules
                                        ▷ $s_0$: Initial state
                                        ▷ $\psi := (\varphi_1$ U $\varphi_2)$
                                        ▷ $maxLength$ := Prespecified maximum path length
2:     currentState := $s_0$;
3:     length := 0;
4:     while currentState $\nvDash \psi \wedge$ length $< maxLength$ do
5:         for rule $\in M$ do
6:             if rule matches currentState then
7:                 currentState := next(rule) ;        ▷ random selection based on rule
8:                 length++;
9:                 break;
10:            end if
11:        end for
12:    end while
13:    if currentState $\vDash \psi$ then return true;
14:    elsereturn false;
15:    end if
16: end procedure
```

the standard PRISM simulator is called. The U2B method can also be bypassed entirely (even for unbounded properties) through a command line switch.

8.2.1 *Phase I*

PRISM's path generation method is described by algorithm 4. This method assumes that some rule from the model will match every state. PRISM ensures this condition by adding self-loops to deadlocked states.

Two primary limitations in this procedure make it unsuitable for its direct use in the *Phase I* of the U2B method in PRISM-U2B. First, the procedure requires the *maxLength* parameter, which effectively defines an upper bound on the path length even for unbounded until path properties. *Phase I* of the U2B method, on the other hand, requires truly unbounded paths in order to calculate an appropriate k_0. Second, the PRISM simulator operates by creating single paths and extending them until the property under consideration is decided, or until the maximum path length is reached. As U2B does not put any restriction on the path length, the above is not a good strategy for random generation of paths in the *Phase I*. For instance, for the property φ_1 U φ_2, if the path being generated is a part of an unconditional loop in the model and satisfies the property $\varphi_1 \wedge \neg\varphi_2$, then the above path generation method will lead to generation of an infinite length path and therefore never terminate. Algorithms proposed in Sections 5 and 6 provide a better strategy for path generation in *Phase I* of `U2B` (and `U2B_P`). This requires that the simulator in `PRISM-U2B` randomly generate and keep track of multiple paths using the last state in the corresponding path (where the number of states corresponds to the *window size* in Algorithms 2 and 3 at a time.

Much of the infrastructure required to implement Algorithms 2 and 3 is already present in `PRISM`. Rather than maintaining a single current state, `PRISM-U2B` maintains a set of current states and switches between them in order to extend the paths evenly. The *maxLength* parameter and the length checking condition are simply removed for *Phase I*.

8.2.2 *Phase II*

In *Phase II*, only k_0-bounded until properties are considered, where k_0 is obtained from *Phase I*. Therefore, for `DTMC` models, the `PRISM` simulator is directly used in *Phase II* to estimate the k_0 bounded until property using N_2 (as prescribed by *Phase II* of `U2B`) samples of paths with length k_0.

8.3 Application to `CTMC` models

As previously described, the U2B method can be applied to unbounded CSL properties on CTMC models through their embedded DTMCs. Bounded properties in CSL are bounded in time rather than number of steps. One step may represent a relatively small or large amount of time, and steps may vary in the amount of time that they represent. Therefore, there is no direct translation from the step bound calculated by the first phase of the U2B method to a time bound in a CSL property.

When verifying CSL properties on CTMCs, the first phase of `PRISM-U2B` is essentially identical. Instead of being generated from a probability distribution, paths are generated from the rates on outgoing transitions. Once a step bound k_0 has been established, the CSL properties are left unmodified. Instead, the internal maximum path length for the default

PRISM simulator is set to k_0. By default, when a path reaches that length, the PRISM simulator prints an error message and does not return a result. For PRISM-U2B, this behavior is modified so that a property is assumed to be false when a path reaches the maximum length.

8.4 Plateau detection

As described in 6.2, there are models and properties for which Algorithm 2 will not terminate. If desired, the user can invoke Algorithm 3 through a command line switch. Algorithm 3 is not deployed by default because it is not known whether the algorithm has found a true plateau without preanalysis of the model.

8.5 PRISM-U2B Usage Description

The interfaces (command line and graphical) of PRISM-U2B follow closely those of PRISM. We proceed by describing various command line options that are added to the existing ones. While in PRISM the statistical model checking can be invoked by

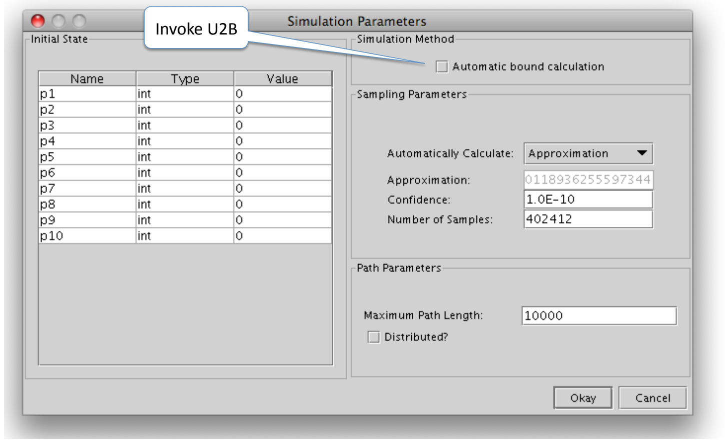
```
prism -sim <model-file> <property-file>
```

the U2B method based statistical model checking can be invoked in PRISM-U2B with an additional qualification of the type of statistical method

```
prism -sim 2 <model-file> <property-file>.
```

PRISM's statistical method can be invoked by setting sim flag to 1. In the event that all input properties are (step or time) bounded until properties, PRISM-U2B directly invokes PRISM's existing sampling technique. PRISM's statistical method allows users to input the error bound, the confidence parameter, and the maximum path lengths. If no such value is specified, PRISM uses the default values of error bound (0.01), confidence parameter (10^{-10}) and maximum path length ($10,000$). In a similar fashion, PRISM-U2B uses default values for the error bounds and the confidence parameters in the two phases of U2B method: $\epsilon_0 = 0.0025$, $\epsilon_1 = 0.0125$, $\epsilon_2 = 0.01$ and $\delta_1 = \delta_2 = 0.005$. PRISM-U2B also allows for user-specified error bounds and confidence parameters for the U2B method. The user can provide specific values for the individual ϵs

(a)



Initial State

Name	Type	Value
p1	int	0
p2	int	0
p3	int	0
p4	int	0
p5	int	0
p6	int	0
p7	int	0
p8	int	0
p9	int	0
p10	int	0

Simulation Method

☐ Automatic bound calculation

Sampling Parameters

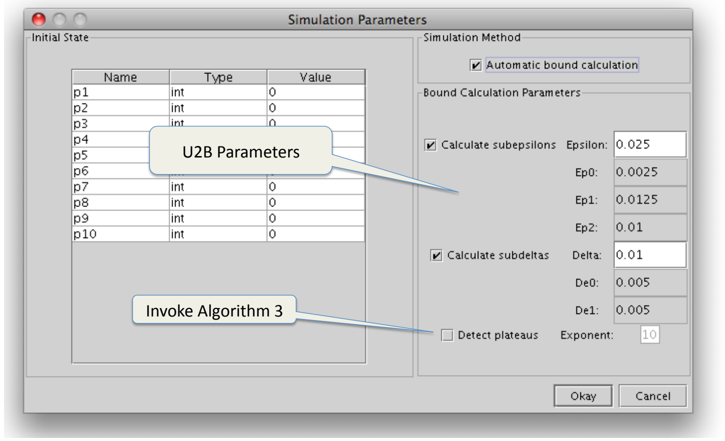
Automatically Calculate:	Approximation
Approximation:	0118936255597344
Confidence:	1.0E-10
Number of Samples:	402412

Path Parameters

Maximum Path Length: 10000

☐ Distributed?

(b)



Initial State

Name	Type	Value
p1	int	0
p2	int	0
p3	int	0
p4		
p5		
p6		
p7	int	0
p8	int	0
p9	int	0
p10	int	0

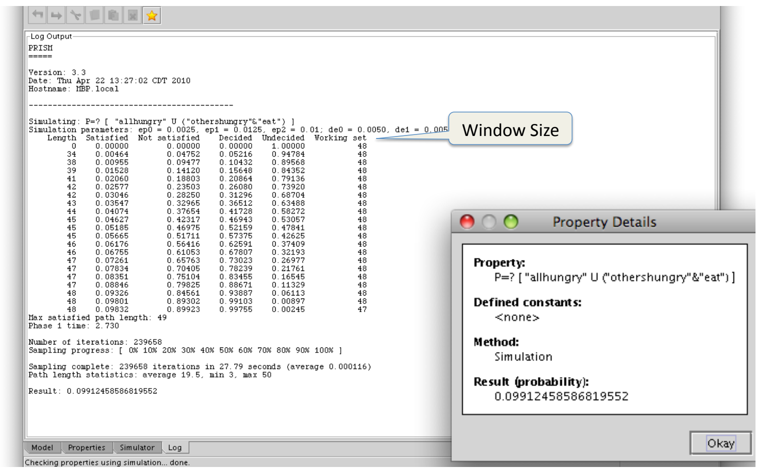
Simulation Method

✓ Automatic bound calculation

Bound Calculation Parameters

✓ Calculate subepsilons	Epsilon:	0.025
	Ep0:	0.0025
	Ep1:	0.0125
	Ep2:	0.01
✓ Calculate subdeltas	Delta:	0.01
	De0:	0.005
	De1:	0.005
☐ Detect plateaus	Exponent:	10

(c)



Log Output

```
PRISM
=====

Version: 3.3
Date: Thu Apr 22 13:27:02 CDT 2010
Hostname: MBP.local

-------------------------------------------

Simulating: P=? [ "allhungry" U ("othershungry"&"eat") ]
Simulation parameters: ep0 = 0.0025, ep1 = 0.0125, ep2 = 0.01; de0 = 0.0050, de1 = 0.005
    Length  Satisfied  Not satisfied    Decided  Undecided  Working set
         0    0.00000        0.00000    0.00000    1.00000           48
        34    0.00464        0.04752    0.05216    0.94784           48
        38    0.00955        0.09477    0.10432    0.89568           48
        39    0.01528        0.14120    0.15648    0.84352           48
        41    0.02060        0.18803    0.20864    0.79136           48
        42    0.02577        0.23503    0.26080    0.73920           48
        42    0.03046        0.28250    0.31296    0.68704           48
        43    0.03547        0.32965    0.36512    0.63488           48
        44    0.04074        0.37654    0.41728    0.58272           48
        45    0.04627        0.42317    0.46943    0.53057           48
        45    0.05185        0.46975    0.52159    0.47841           48
        45    0.05665        0.51711    0.57375    0.42625           48
        46    0.06176        0.56416    0.62591    0.37409           48
        46    0.06755        0.61053    0.67807    0.32193           48
        47    0.07261        0.65763    0.73023    0.26977           48
        47    0.07834        0.70405    0.78239    0.21761           48
        47    0.08351        0.75104    0.83455    0.16545           48
        47    0.08846        0.79825    0.88671    0.11329           48
        48    0.09326        0.84561    0.93887    0.06113           48
        48    0.09801        0.89302    0.99103    0.00897           48
        48    0.09832        0.89923    0.99755    0.00245           47
Max satisfied path length: 49
Phase 1 time: 2.730

Number of iterations: 239658
Sampling progress: [ 0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100% ]

Sampling complete: 239658 iterations in 27.79 seconds (average 0.000116)
Path length statistics: average 19.5, min 3, max 50

Result: 0.09912458586819552
```

Checking properties using simulation... done.

Property:
P=? ["allhungry" U ("othershungry"&"eat")]

Defined constants:
<none>

Method:
Simulation

Result (probability):
0.09912458586819552

Figure 8.2 PRISM-U2B GUI

(and/or δs) or can provide a single global ϵ (and/or a global δ). In the latter case, the global error bound is partitioned using the default factors: $\epsilon_0 = \frac{\epsilon}{10}, \epsilon_1 = \frac{\epsilon}{2}, \epsilon_2 = 2\frac{\epsilon}{5}$, and the global confidence parameter is partitioned into two equal halves for δ_1 and δ_2. For instance,

```
prism -sim 2 -simepsilon 0.002 0.001 0.01 -simdelta 0.0001
                                     <model-file> <property-file>
```

This provides as input $\epsilon_0 = 0.002$, $\epsilon_1 = 0.001$, $\epsilon_2 = 0.01$, and a global $\delta = 0.0001$. In the above, PRISM-U2B uses Algorithm 2 for computing k_0 in *Phase I*. If the user chooses to use method U2B_P, he or she can invoke

```
prism -sim 2 -simpoly 3 <model-file> <property-file>
```

The value of the switch simpoly controls the length of plateau (as discussed in Section 6.2) and ensures the termination of statistical sampling based method. Inputs can be supplied in a similar fashion when PRISM-U2B is invoked via the graphical user interface (Figure 8.2). The tool, PRISM-U2B, along with its source code, sample models, and documentation, is available from Jennings et al. (2010).

CHAPTER 9 Experimental Evaluation

We evaluate the proposed U2B method and its realization in PRISM-U2B using a number of case studies available in the PRISM example suite. The objective of this empirical study is to show the effectiveness of the U2B method with respect to (a) precision of estimate, (b) computation time, and (c) memory usage. We proceed by describing in brief the various case studies in Section 9.1 followed by a detailed analysis of empirical results in Sections 9.2 and 9.3.

9.1 Case Studies

As noted before, PRISM-U2B re-uses PRISM's input specification language and language parsers, and therefore, our experiments directly use several case studies from the PRISM example suite. PRISM (and by extension PRISM-U2B) allows for a specific type of PCTL and CSL property syntax, which essentially *queries* the probability with which certain path property is satisfied by the system under consideration. The syntax for such a query is `P=?[path-property]`. The result of such a query is the probability with which the paths from the start state of the system satisfies `path-property`. In the following, we will discuss case studies and consider several property queries. The results obtained by applying PRISM's numerical method (whenever possible) will be used to evaluate and compare the precision of results obtained by applying PRISM's statistical method and the U2B method of PRISM-U2B.

9.1.1 Randomized Dining Philosopher

This is a DTMC model represented a probabilistic variation of the standard dining philosopher protocol. The model is analyzed against a PCTL until property query to obtain the probability that the first philosopher is able to eat before any other philosophers. That is,

```
P=?[ "allhungry" U ("othershungry" & "eat") ]
```

where `allhungry` denotes that none of the philosophers have eaten, `othershungry` denotes that all but the first philosopher have not eaten and `eat` denotes that the first philosopher is able to eat. Experiments are conducted by varying the number of philosophers in the model. Available at: Jennings et al. (2010).

9.1.2 Simple Queue

This is a `DTMC` model of a queue in which requests are either received or serviced with a particular probability distribution. If too many requests are received before they can be serviced, then there is an *overflow* of requests. The property considered computes the probability with which the queue eventually reaches such a state.

```
P=?[ true U "overflow"]
```

Experiments are conducted by varying the size of the buffer (queue states) in the model. Available at: Jennings et al. (2010).

9.1.3 IPv4 Zeroconf Protocol

This a simplified `DTMC` model created by Sen et al. (2005) representing the IPv4 Zeroconf Protocol. Similar to the overflow state in the queue model, it has an *error* state; it is not desirable for the system executing the protocol to be in such a state. The property query used in our experiments is

```
P=?[ true U "error"]
```

As in the queue, experiments are conducted by varying the number of states (denoting the number of intermediate nodes in the network) in the Zeroconf protocol.

9.1.4 Crowds Protocol

This is a DTMC model of a protocol from Reiter and Rabin (1998) for anonymous Web browsing. The protocol is intended to minimize the chance that eavesdropping adversaries will be effective in identifying the individual users by spying on network traffic. Experiments are done with models containing 50, 100 and 200 hosts (size of the crowd) and four adversaries. The property query involves estimating the probability that the identify of an user is divulged, which happens when at least two adversaries observe (agree on) the user's identity.

```
P=?[true U (observe0>1)]
```

In the above, `observe[`i`]` denotes the number of adversaries that have observed the user i.
Available at: `http://www.prismmodelchecker.org/casestudies/crowds.php`.

9.1.5 Broadcast Protocol

This is a DTMC model of a simple broadcast protocol based on *gossiping*, where each node in the network forwards the message it receives to its neighbors with a pre-specified probability. Two variations of the protocol are considered. In the *synchronous, no-collision* variation, the nodes send and receive messages simultaneously over independent channels (thereby ensuring freedom from collision). The *synchronous, lossy* variation, on the other hand, assumes that the neighboring nodes share a channel, which may result in collisions and message loss. The property query of interest involves computing the probability with which certain nodes (e.g., nodes 1 and 2) receive the broadcast message.

`P=?[ true U (active1=0) ]` and `P=?[ true U (active2=0) ]`

In the above `active[i]` is equal to 0 when a node goes to sleep after receiving (and possibly forwarding) message.
Available at: `http://www.prismmodelchecker.org/casestudies/prob_broadcast.php`.

9.1.6 EGL Contract Signing Protocol

EGL Contract signing protocol describes a method by which two parties can reach an agreement by exchanging their respective secrets. The protocol is said to be *fair* if and only if the exchange of messages following the protocol guarantees that either both participants obtain the other's secret or none do. A DTMC model of the EGL contract signing protocol is considered. A property query is considered to demonstrate that the protocol is flawed with respect to fairness. That is, the property captures the fact that one participant gets access to the other party's secret without communicating its own secrets.

```
P=?[ true U (!"kA" & "kB") ]
```

In the above, kA and kB denotes the states where participants A and B *have knowledge* of the other's secret.
Available at: http://www.prismmodelchecker.org/casestudies/contract_egl.php

9.1.7 Simple Dice Protocol

This is a DTMC model where the possible outcomes of rolling a fair die are captured using multiple flips of a fair coin. The result of the property query against which the model is analyzed is the probability of obtaining a specific roll of the die. For instance,

```
P=?[true U ("s=7" & "d=6")]
```

is a PCTL property querying the probability of obtaining a roll of 6 on a single die.
Available at: http://www.prismmodelchecker.org/casestudies/dice.php

9.1.8 Embedded Control System

This is a CTMC model of an embedded data collection system with built-in redundancy. The controller ensures that the system shuts down if any module in the system is down (due to possible malfunction) for a certain number of cycles (e.g., $20,000$). The property queries the probability that the cause of shut down is attributed to a sensor failure. It is represented in PCTL query as follows.

```
P=?[ !"down" U "fail_sensors" ]
```

Available at: `http://www.prismmodelchecker.org/casestudies/embedded.php`

9.1.9 Cyclic Server Polling System

This is a `CTMC` model of a polling system consisting of multiple workstations that are being served by a single server. The `CSL` property that is used for our evaluation is

```
P=?[ !(s=2 & a=1) U (s=1 & a=1) ]
```

which queries the probability with which the first workstation in the system is served before the second one. Experiments are conducted by varying the number of workstations in the system.
Available at: `http://www.prismmodelchecker.org/casestudies/polling.php`

9.2 Precision, Computation Time and Memory Usage

The experimental evaluation is based on the illustrative example discussed in Chapter 1 and the case studies described in Section 9.1. All experiments are conducted on Red Hat Enterprise Linux 5.1 running on Intel Core 2 Duo 3GHz CPU and 2GB memory. The results for `PRISM-U2B` are obtained by using the Algorithm 2 as all the examples considered from the `PRISM` example suite satisfy the condition $\lim_{k\to\infty} \mathsf{P}(s,\psi_k) \geq 1-\epsilon_0$ as required for the termination of *Phase I* in the `U2B` method.

9.2.1 Precision & Computation Time

We show that the estimate obtained using `U2B` method is as precise as the statistical method provided by the `PRISM` tool given the error bounds and confidence parameters. Furthermore, the `U2B` method is as efficient as (and typically about 1.5 times faster than) `PRISM`'s statistical method.

Table 9.1 provides a summary of our results. For each of the methods, the table presents the probability and its estimates, and the computation time. For the statistical methods, the

Model	Variations	PRISM					PRISM-U2B			
		Numerical		Statistical			Phase I		Phase II	
		p	Time	$\hat{p}$	$\bar{k}$	Time	k_0	Time	$\hat{p}$	Time
Example (Figure. 1.1)		No result		No result			3343	**7.74**	0.6601	**88.74**
Dining philosopher	# philos: 10	0.1	5.98	0.0999	75	**29.21**	48	**1.5**	0.0997	**16.85**
	# philos: 20	No result		0.0497	103	**89.6**	68	**4.31**	0.0496	**51.97**
	# philos: 40	No result		0.0247	141	**252.49**	98	**11.76**	0.0246	**146.47**
	# philos: 60	No result		0.0163	176	**469.29**	123	**21.75**	0.0166	**271.69**
	# philos: 100	No result		0.0099	227	**1044.12**	164	**48.29**	0.0097	**606.7**
Queue	$n : 80, q : 0.9, r : 0.5$	0.1022	< 1	0.1067	20156	**214.71**	8066	**9.92**	0.105	**126.27**
	$n : 100, q : 0.9, r : 0.5$	0.0832	< 1	0.0871	29906	**275.2**	12083	**12.62**	0.0862	**162.16**
	$n : 140, q : 0.9, r : 0.5$	0.0607	< 1	0.0638	55523	**395.55**	20774	**18.19**	0.0627	**235.46**
	$n : 180, q : 0.9, r : 0.5$	0.0477	< 1	0.0502	83910	**507.59**	32336	**23.75**	0.0491	**303.97**
	$n : 200, q : 0.9, r : 0.5$	0.0431	< 1	No result at $\bar{k} = 10^6$			38928	**26.45**	0.0444	**339.01**
Zeroconf	$n : 10, q : 0.9, r : 0.9$	0.8098	< 1	0.8118	99	**4.18**	44	< 1	0.8098	**2.27**
	$n : 20, q : 0.9, r : 0.9$	0.5975	< 1	0.6015	360	**11.16**	173	< 1	0.5998	**6.54**
	$n : 60, q : 0.9, r : 0.9$	0.0215	< 1	0.0222	1469	**30.72**	636	**1.57**	0.0221	**17.98**
	$n : 80, q : 0.9, r : 0.9$	0.0027	< 1	0.0027	1483	**31.58**	670	**1.61**	0.0028	**18.43**
Crowds	$n : 50$	0.0483	175.42	0.0483	220	**73.88**	135	**3.68**	0.0484	**45.16**
	$n : 100$	No result		0.046	218	**135.85**	135	**6.79**	0.0462	**83.97**
	$n : 200$	No result		0.045	219	**249.39**	135	**13.03**	0.045	**163.16**
Broadcast Protocol	Synch. no Collision	0.8	< 1	0.8001	3	**34.83**	3	**1.79**	0.8003	**19.98**
		0.7324	< 1	0.7329	8	**45.14**	8	**2.27**	0.7326	**26.02**
	Synch. collision, lossy	0.5815	9.3	0.5814	20	**163.26**	20	**7.89**	0.581	**94.95**
		0.3462	9.61	0.3465	22	**225.99**	20	**10.84**	0.3463	**131.27**
Contract Signing		1.0	< 1	1.0	36	**60.99**	36	**3.12**	1.0	**33.88**
Dice		0.1667	< 1	0.1646	23	**2.39**	13	< 1	0.1642	**1.21**
ECS	$MAX\_COUNT : 20,000$	0.7555	24568.49	No result at $\bar{k} = 10^6$			122231	**1520.05**	0.7532	**25029.29**
Polling	$N : 2$	0.5	< 1	0.5	2581	**178.76**	1195	**8.09**	0.4983	**103.73**
	$N : 20$	0.538	8713.28	0.5384	9714	**3847.5**	4261	**179.05**	0.5377	**2230.01**

Table 9.1 Summary of Results: `PRISM` vs. `PRISM-U2B`

bounds on the sample path lengths are also reported: $\bar{k}$ denotes the maximum path length considered by `PRISM`'s statistical method and k_0 denotes the maximum path length considered by `PRISM-U2B` (the bound obtained automatically from *Phase I* of `U2B`).

For the illustrative example, `PRISM` fails to compute any result, while `PRISM-U2B` successfully terminates with a good estimate. For the randomized dining philosopher example, `PRISM`'s numerical method fails to compute any result when the number of philosophers is ≥ 12 due to prohibitively large state-space of the `DTMC` model (details in Section 9.2.2), whereas `PRISM`'s statistical method and `U2B` successfully compute the estimate. For the simplified queue model, `PRISM`'s numerical method and statistical method both require updates to the default value for maximum number of Jacobi iterations or the maximum simulation path length. The `U2B` method does not require any such user guidance.

Observe that for the rest of the case studies, `PRISM-U2B` typically outperforms `PRISM`'s statistical method (timing results shown in bold). While both `U2B` in *Phase II* and `PRISM`'s statistical method use the same sampling technique, the former uses a much smaller sample path length bound (k_0 obtained via *Phase I*) compared to `PRISM`'s statistical method (which uses $\bar{k}$; see Table 9.1). Recall that the default value for sample path length in `PRISM`'s statistical method is $10,000$ and may require the user to specify a greater value for some examples. The `U2B` method uses *Phase I* to compute a model dependent path length bound k_0. The gain in computation time achieved by using k_0 in *Phase II* of `U2B` instead of the sample path lengths used in `PRISM`'s statistical method surpasses the loss in computation time spent computing k_0 in *Phase I*.

9.2.2 Memory Usage

Another important consideration is the memory consumed by each of the methods. Figure 9.1 is a graph of the memory usage (in terms of KB) for the case studies for each of the methods. As all the methods are realized in the `PRISM` model checking engine, each incurs a minimal overhead of $\sim$ 30MB. The memory usage is comparable for all the methods for models that have small state-space. As expected, for models in which the state-space is large (e.g.,

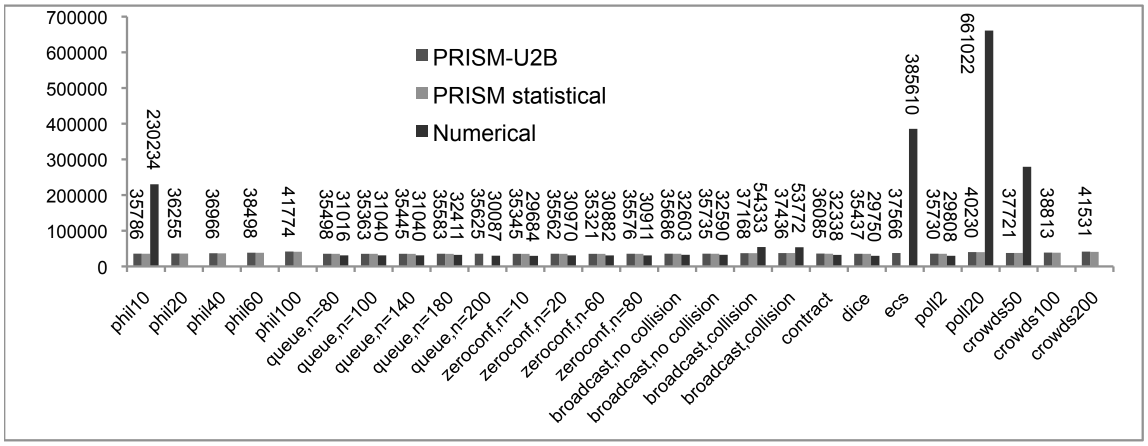


Figure 9.1 Memory usage (in KB) for Case Studies.

randomized dining philosopher and embedded control system), PRISM's numerical method uses much more memory than the statistical methods (in PRISM and PRISM-U2B). Figure 9.2 shows the increase in the memory usage by PRISM's numerical method with the increase in the number of philosophers in the randomized dining philosopher case study. PRISM's numerical method fails due to the large memory requirement when the number of philosophers is greater than 11. Observe that the statistical methods do not suffer from such an increase in the memory usage with the increase the state-space of the model.

It is worth mentioning that U2B method in general is expected to use more memory than that used by the statistical method of PRISM as U2B method stores $N_1\epsilon_0$ states for the *Phase I* computation (for $\epsilon_0 = 0.0025$ and $\delta = 0.001$, $N_1\epsilon_0 = 48$).

9.2.3 Summary of results

The results empirically show that the U2B method as implemented in PRISM-U2B is (a) as time-efficient as PRISM's statistical method (and in many cases outperforms PRISM's statistical method), (b) as precise as PRISM's statistical method, and (c) successfully computes results *automatically* for cases where PRISM fails or requires user guidance (to increase the maximum Jacobi iterations for the numerical method or the maximum sample path length for the numerical method). Finally, as PRISM-U2B utilizes the model specification language, graphical

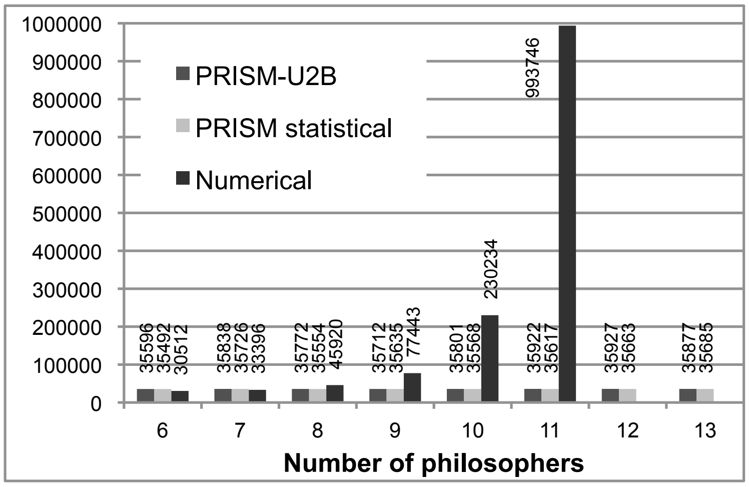


Figure 9.2 Memory usage for (in KB) Randomized Dining Philosopher.

user interface, and command-line interface of PRISM, it enjoys (as does PRISM) a high rating among all the probabilistic model checkers in terms of usability, as determined by Jansen et al. (2007). In short, PRISM-U2B broadens the scope of application of approximate probabilistic model checking based on statistical methods; it is not only more efficient and effective than one of the most widely used statistical method as implemented in PRISM but also has been proven (theoretically and empirically) to be applicable in case studies where PRISM fails.

9.3 Comparison with the MRMC tool

Katoen and Zapreev have performed extensive comparison of the statistical model checking tools MRMC (based on Zapreev (2008)), VESTA (based on Sen et al. (2005)), and Ymer (based on Younes and Simmons (2002)); and concluded that MRMC is the fastest among the three. The version of Ymer used in this study did not allow verification of (time) unbounded until properties, while VESTA and MRMC do support verification of these properties. Recall from Chapter 3 that the statistical method in VESTA suffers from two main drawbacks which restrict its correct application in models with loops (see Younes and Simmons (2006); He et al. (2010) for details). Furthermore, we were unable to obtain the tool from the authors (personal communication with Koushik Sen dated: 11/04/2010). While the tool MRMC does not have any such restriction, it requires prior analysis of the model transition structure to correctly verify unbounded until properties even when using the sampling based method. In the following, we compare MRMC and PRISM-U2B, and discuss their distinguishing aspects.

Markov Reward Model Checker supports verification of probabilistic temporal logic properties (PCTL, CSL, etc.) against different types of probabilistic models (DTMC, CTMC, etc.) using both numerical and statistical sampling based methods. MRMC's statistical method utilizes confidence interval based estimation to control the error in inferring whether the null hypothesis is correctly rejected or not. As this method utilizes both confidence interval based statistical estimation (Section 3.1.3) and hypothesis testing (Section 3.1.1), we present below an overview of the technique to clearly explain the user-specified inputs that are necessary for correct application of MRMC's statistical method.

Recall that PRISM and PRISM-U2B are meant to compute the probability of satisfying a given path property within pre-specified error margin and with a certain confidence bound. In contrast, probabilistic queries in MRMC are of the form $\mathtt{P}_{\bowtie r}(\psi)$, where $\bowtie \in \{<, >, \leq, \geq\}$ and ψ is a path property. Therefore, the result or output of MRMC is boolean. More precisely, MRMC uses the statistical estimate $\hat{p}$ using the confidence interval and infers that the relation $\hat{p} \bowtie r$ holds when the confidence interval does not include r. For instance, if $\bowtie$ is equal to $\geq$, then the method can provide a definitive answer (true) when lower bound of the confidence interval is greater than r, and a definitive answer (false) when the upper bound of the confidence interval is less than r. However, due to the probabilistic nature of the confidence interval, it is necessary to assume that r is at least ξ distance from the true probability (where the indifference width 2ξ; see Section 3.1.1) and that the confidence interval is tighter than ξ. In short, MRMC draws from techniques used in hypothesis testing (as applied in probabilistic model checking) as well as confidence interval based statistical estimation. MRMC therefore requires the following user inputs: confidence level δ for estimation using the confidence interval and $2\xi'$, the confidence interval (such that $\xi' < \xi$). That is, ξ' corresponds to the ϵ used in PRISM-U2B.

Zapreev (2008) proposes a technique (realized in the tool MRMC) that deals with unbounded until properties by pre-analyzing the model under consideration. More specifically, for the property φ_1 U φ_2, the sampling based method in MRMC first identifies (and discards) the states that never lead to any state satisfying φ_1 U φ_2. Such reachability analysis requires complete knowledge of the model transition structure, which may not be possible when the model state-space is prohibitively large or the model of the system is not available (only samples simulations can be generated as needed).

Another important difference between MRMC and both PRISM and PRISM-U2B (which possibly stems from MRMC's requirement to pre-analyze the model) is that MRMC reads/loads the entire transition matrix of the probabilistic model for its sampling based method. Therefore, the memory usage of sampling based method in MRMC is similar to that of numerical methods (e.g., in PRISM's numerical method modulo the differences that stem from a different implementation framework). This significantly reduces the applicability of MRMC's sampling based method for large systems. In fact, for several examples in Table 9.1, we cannot obtain results using MRMC as it leads to an out-of-memory error.

MRMC does not have a modeling language of it's own. The input files are simply the full transition matrix. PRISM provides the option of exporting MRMC-formatted files for use in MRMC; the authors suggest using this method for generating models. For several of the models in Table 9.1, PRISM is unable to export the files because of the large size of the models.

Table 9.2 summarizes the results of our experiments with MRMC. We have used an identical $\delta = 0.01$ (confidence parameter) and $\epsilon = \xi' = 0.025$ for the experiments. As MRMC tests whether the estimate is related to a given value r using $\bowtie$ relation and requires that the true p be $\xi > \xi'$ distance away from r, we have set r to be equal to $p + \xi$ where $\xi = 0.026 > \xi'$. This allows MRMC to provide definitive answers (true or false) in all the cases. If ξ' is chosen to be greater than ξ, MRMC fails to compute a definitive answer. Note that no such assumption is required for the validity of our method as implemented in PRISM-U2B.

In the table, the column p corresponds to the "actual" probability of satisfying the property being considered. It is either computed manually, computed by PRISM's numerical method, or estimated by PRISM's statistical method. The query input to MRMC tests whether the estimated probability $\bowtie$ $p \pm 0.026$. The boundaries of the confidence interval estimated by MRMC are

Model	Variations	p	MRMC			PRISM-U2B			
			Lower bound	Upper bound	Time	Phase I		Phase II	
						k_0	Time	$\hat{p}$	Time
Example (Figure 1.1)		0.66	Cannot load			3343	7.74	0.6601	88.74
Dining Philosopher	# philos: 10	0.1	Cannot load			48	1.5	0.0997	16.85
Queue	$n: 80, q: 0.9, r: 0.5$	0.1022	0.0929	0.1047	< 1	8066	9.92	0.105	126.27
	$n: 100, q: 0.9, r: 0.5$	0.0832	0.0737	0.0901	< 1	12083	12.62	0.0862	162.16
	$n: 140, q: 0.9, r: 0.5$	0.0607	0.0416	0.0784	< 1	20774	18.19	0.0627	235.46
	$n: 180, q: 0.9, r: 0.5$	0.0477	0.0273	0.0714	< 1	32336	23.75	0.0491	303.97
	$n: 200, q: 0.9, r: 0.5$	0.0431	0.0230	0.0693	< 1	38928	26.45	0.0444	339.01
Zeroconf	$n: 10, q: 0.9, r: 0.9$	0.8098	0.8070	0.8152	< 1	44	< 1	0.8098	2.27
	$n: 20, q: 0.9, r: 0.9$	0.5975	0.5849	0.6072	< 1	173	< 1	0.5998	6.54
	$n: 60, q: 0.9, r: 0.9$	0.0215	0.0180	0.0230	< 1	636	1.57	0.0221	17.98
	$n: 80, q: 0.9, r: 0.9$	0.0027	0.0006	0.0047	< 1	670	1.61	0.0028	18.43
Crowds	$n: 50$	0.0483	Cannot model check			135	3.68	0.0484	45.16
	$n: 100$	0.0450	PRISM cannot export			135	6.79	0.0462	83.97
Broadcast Protocol	Synch. no collision	0.800	0.7969	0.8108	< 1	3	1.79	0.8003	19.98
		0.7324	0.7247	0.7478	< 1	8	2.27	0.7326	26.02
	Synch. collision, lossy	0.5815	0.5716	0.5919	< 1	20	7.89	0.5810	94.95
		0.3462	0.3365	0.3602	< 1	20	10.84	0.3463	131.27
Contract Signing		1.0	Result with no simulation			36	3.12	1.0	33.88
ECS	MAX_COUNT: 20,000	0.7555	PRISM cannot export			122231	1520.05	0.7532	25029.29
Poll	$N: 2$	0.500	0.4894	0.5077	< 1	1195	8.09	0.4983	103.73
	$N: 20$	0.0538	PRISM cannot export			4261	179.05	0.5377	2230.01

Table 9.2 Summary of results: MRMC vs. PRISM-U2B

presented in the columns Lower bound and Upper bound. MRMC produces results faster than PRISM-U2B in all the cases where it is successful in computing the result. This is because (as noted above) MRMC analyzes the model before performing sampling based confidence interval estimation. That is, MRMC does not deploy a pure sampling based statistical method. This is advantageous for smaller models whose reachability can be performed. In fact, in some cases, sampling may not be even necessary. For instance, for the contract signing protocol, MRMC produces a definitive answer (infers that the property is satisfied) without using any samples. However, if such pre-analysis of the model is not possible, then MRMC fails. This happens for several examples when the transition matrix is too large to be exported from PRISM to MRMC (e.g. the example in Figure 1.1, Crowds protocol of size ≥ 100, Embedded Control System, Cyclic Server Polling System of size 20) or when the reachability analysis fails in MRMC (Dining philosopher protocol with ≥ 10 philosophers, Crowds protocol of size 50).

In summary, the following aspects distinguish PRISM-U2B from MRMC:

1. MRMC relies on reachability analysis in addition to samples to compute the results. PRISM-U2B deploys a pure sampling based method. For the case studies for which MRMC can perform reachability analysis, MRMC computes verification result faster than PRISM-U2B.

2. MRMC fails for models where the transition matrix is too large for loading or for performing reachability analysis. PRISM-U2B can be used for comparatively larger models.

3. MRMC requires that the user-input ξ' (the parameter specifying the confidence interval size) be smaller than the distance (half of indifference width) of r from the true p in the query of the form $\mathtt{P}_{\bowtie r}(\psi)$ where $\bowtie$ belongs to $\{<, \leq, >, \geq\}$. There is no such requirement for PRISM-U2B. While MRMC may not be able to compute a definitive answer when the confidence interval size is not less than the indifference width, the U2B algorithm in PRISM-U2B may not always terminate (Chapter 6). None of the case studies, however, led to the non-termination of the U2B algorithm in PRISM-U2B.

CHAPTER 10 Conclusion

10.1 Conclusion

We have presented U2B, an approximate probabilistic model checking method based on statistical sampling. The method does not require any prior information regarding the structure of the model being verified and therefore can be used in a setting where only sample simulations of the model can be generated. The method can be applied for verifying untimed properties in both DTMC and CTMC models. We have proved the correctness of the U2B method and have discussed the optimized realization of the U2B method. We have also explored a class of examples where PRISM's statistical method will not be able to estimate a result and the U2B method will suffer from non-termination. We have explained the cause of such problems in terms of the notion of a "plateau" in the D-Graph. We have developed U2B_P, a heuristic-based plateau-detection method, to deal with the non-termination problem. Finally, we have incorporated the U2B and U2B_P methods in the PRISM tool to develop PRISM-U2B, and conducted a detailed experimental study using different probabilistic models. The experiments show the effectiveness of the tool PRISM-U2B in terms of precision, computation time and broader applicability (for large probabilistic models).

10.2 Future work

10.2.1 Implementation in PRISM version 4

PRISM version 4 introduces significant changes to the simulator architecture. The new version implements a frontend/backend approach such that different probabilistic methods can be "plugged in" in order to perform model checking. This allows any of the different model checking methods described in Chapter 3 to be implemented within PRISM. Currently the methods described by Herault et al. (2004) and Younes and Simmons (2002) are implemented.

This new architecture does not directly ease the implementation of U2B, because all of the approaches described in Chapter 3 are implemented by checking one simulation path at a time, not multiple paths as required by U2B. Therefore, U2B will be implemented as a layer above the new PRISM simulator. This format will also allow *Phase II* to utilize any of the implemented methods, rather than only the approach implemented in PRISM version 3.

The basic U2B method has already been ported to PRISM 4. A more thorough investigation would involve experimental comparison amongst PRISM 3, PRISM 4, PRISM-U2B as a modification to PRISM 3, and PRISM-U2B as a modification to PRISM 4.

10.2.2 Termination of *Phase I*

The current heuristic for termination of *Phase I* is unsatisfactory in the sense that it requires the user either to have knowledge of the model (and thus know that when a true plateau will be reached); or guess when a true plateau has been reached. Although some guidance about the length of a plateau is provided by the current implementation, in general the user will not know with certainty when it is appropriate to deploy U2B_P.

As part of future work, we plan to study the feasibility of identifying and pre-computing only the *necessary* information (regarding the model transition structure) required to guarantee the termination of *Phase I* of the U2B method. If this computation can be made less complex than numerical model checking, it may be possible to remove the user-specified parameter from U2B_P.

10.2.3 Markov decision processes

We also plan to develop statistical sampling based methods to verify untimed, unbounded path properties of Markov Decision Processes (MDP). An MDP is a probabilistic model which includes both probabilistic and non-deterministic choices in its transition system. The verification objective for an MDP is to compute the maximum or the minimum probability with which a state satisfies a given path property. Such computation requires identifying a strategy for selecting next states at non-deterministic choice points, where each strategy results in a DTMC embedded in the corresponding MDP.

It is known that for any MDP, there exist memoryless schedulers that correspond to the minimum and maximum probabilities. A naive method would be to apply U2B for every possible memoryless scheduler and then select the optimum. However, the number of memoryless schedulers is $O(|S|^n)$, where n is the maximum number of non-deterministic choices at any state in the model, and $|S|$ is the number of states.

Work is underway on a method that attempts to sample some subset of the memoryless schedulers and bound the probability of selecting a scheduler that is less than optimal. Both the selection of the scheduler and the model checking of the underlying DTMC could be probabilistic, thus avoiding the problem of state space explosion.

BIBLIOGRAPHY

Alur, R. and Henzinger, T. (1999). Reactive modules. *Formal Methods in System Design*, 15(1):7–48.

Aziz, A., Sanwal, K., Singhal, V., and Brayton, R. (1996). Verifying continuous time markov chains. In *International Conference on Computer Aided Verification*, volume 1102.

Aziz, A., Sanwal, K., Singhal, V., and Brayton, R. (2000). Model checking continuous time markov chains. *ACM Transactions on Computational Logic*, 1(1):162–170.

Baier, C., Haverkort, B., Hermanns, H., and Katoen, J.-P. (2003). Model-checking algorithms for continuous-time markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541.

Bianco, A. and de Alfaro, L. (1995). Model checking of probabilistic and nondeterministic systems. In *Foundations of Software Technology and Theoretical Computer Science*, volume 1026.

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

Duflot, M., Kwiatkowska, M., Norman, G., and Parker, D. (2006). A formal analysis of bluetooth device discovery. *Intl. Journal on Software Tools for Technology Transfer*, 8:621 – 632.

Forsythe, G. and Leibler, R. (1950). Matrix inversion by a Monte Carlo method. *Mathematical Tables and Other Aids to Computation*, 4(31):127–129.

Grosu, R. and Smolka, S. A. (2005). Monte carlo model checking. In *International Conference on Tools and Algorithms for the Contruction and Analysis of Systems*, pages 271–286.

Grubbs, F. (1949). On designing single sampling inspection plans. *The Annals of Mathematical Statistics*, 20(2):242–256.

Grunske, L. (2008). Specification patterns for probabilistic quality properties. In *International Conference on Software Engineering*, pages 31–40.

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

He, R., Jennings, P., Basu, S., Ghosh, A., and Wu, H. (2010). A bounded statistical approach for model checking of unbounded until properties. In *IEEE/ACM International Conference on Automated Software Engineering*, pages 225–234.

Herault, T., Lassaigne, R., Magniette, F., and Peyronnet, S. (2004). Approximate probabilistic model checking. In *5th Intl. Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 2937. Springer.

Herault, T., Lassaigne, R., and Peyronnet, S. (2006). APMC 3.0: Approximate verification of discrete and continuous time Markov chains.

Hinton, A., Kwiatkowska, M., Norman, G., and Parker, D. (2006). PRISM: A tool for automatic verification of probabilistic systems. In *12th Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58.

Jansen, D. N., Katoen, J.-P., Oldenkamp, M., Stoelinga, M., and Zapreev, I. (2007). How fast and fat is your probabilistic model checker. In *Haifa Verification Coference*, pages 69–85.

Jennings, P., Basu, S., and Ghosh, A. (2010). Two-phase pmck. Available at `http://www.cs.iastate.edu/~poj/prism-u2b`.

Katoen, J.-P. and Zapreev, I. (2009). Simulation-based ctmc model checking. In *International Conference on the Quantitative Evaluation of Systems*.

Kwiatkowska, M., Norman, G., and Parker, D. (2008). Using probabilistic model checking in systems biology. *ACM SIGMETRICS Perf. Eval. Review*, 35:14–21.

Legay, A. and Delahaye, B. (2010). Statistical model checking : An overview. *CoRR*, abs/1005.1327.

Massart, P. (1990). The tight constant in the Dvoretzky-Kiefer-Wolfowitz inequality. *Annals of Probability*, 18:1269–1283.

Matsumoto, M. and Nishimura, T. (1998). Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8:3–30.

MRMC (2010). Markov reward model checker. Available from `http://www.mrmc-tool.org/trac`.

Norman, G. and Shmatikov, V. (2006). Analysis of probabilistic contract signing. *Journal of Computer Security*, 14:561–589.

Propp, J. and Wilson, D. (1996). Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random structures and Algorithms*, 9(1-2):223–252.

Rabih, D. and Pekergin, N. (2009). Statistical model checking using perfect simulation. In *International Symposium on Automated Technology for Verification and Analysis*, pages 120–134.

Rabih, D. and Pekergin, N. (2011). Perfect simulator 2.

Reiter, M. K. and Rabin, A. D. (1998). Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92.

Roy, A. and Gopinath, K. (2005). Improved probabilistic models for 802.11 protocol verification. In *14th Intl. Conf. on Computer Aided Verification*.

Sen, K., Viswanathan, M., and Agha, G. (2005). On statistical model checking of stochastic systems. In *14th Intl. Conf. on Computer Aided Verification*, volume 3576. Springer.

Wald, A. (1945). Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16(2).

Younes, H., Clarke, E., and Zuliani, P. (2011). Statistical Verification of Probabilistic Properties with Unbounded Until. *Formal Methods: Foundations and Applications*, pages 144–160.

Younes, H. L., Kwiatkowska, M., Norman, G., and Parker, D. (2006). Numerical vs. statistical probabilistic model checking. *Intl. Journal on Software Tools for Technology Transfer*, 8(3).

Younes, H. L. S. (2005a). *Verification and Planning for Stochastic Processes with Asynchronous Events*. PhD dissertation, Carnegie Mellon University.

Younes, H. L. S. (2005b). Ymer: A statistical model checker. In *International Conference on Computer Aided Verification*.

Younes, H. L. S. and Simmons, R. G. (2002). Probabilistic verification of discrete event systems using acceptance sampling. In *14th Intl. Conf. on Computer Aided Verification*, volume 2404. Springer.

Younes, H. L. S. and Simmons, R. G. (2006). Statistical probabilistic model checking with a focus on time-bounded properties. *Information and Computation*, 204(9).

Zapreev, I. S. (2008). *Model Checking Markov Chains: Techniques and Tools*. PhD dissertation, University of Twente, The Netherlands.