# Ymer: A Statistical Model Checker*

Håkan L.S. Younes

Computer Science Department, Carnegie Mellon University,
Pittsburgh, PA 15213, USA

**Abstract.** We present Ymer, a tool for verifying probabilistic transient properties of stochastic discrete event systems. Ymer implements both statistical and numerical model checking techniques. We focus on two features of Ymer: distributed acceptance sampling and statistical model checking of nested probabilistic statements.

## 1 Introduction

Ymer is a tool for verifying probabilistic transient properties of stochastic discrete event systems. Properties are expressed using the logics PCTL [2] and CSL [1]. For example, the CSL property $\neg\mathcal{P}_{\geq 0.01}[\top\ \mathcal{U}^{[0,15.07]}\ \mathit{faulty}{=}n]$ asserts that the probability of $n$ servers becoming faulty within 15.07 seconds is less than 0.01. In general, $\Phi\ \mathcal{U}^{[\tau_1,\tau_2]}\ \Psi$ is a path formula and is evaluated over execution paths for a stochastic system. The formula $\mathcal{P}_{\geq\theta}[\varphi]$, where $\varphi$ is a path formula, holds if and only if the probability measure of the set of paths that satisfy $\varphi$ is at least $\theta$. To solve CSL model checking problems, one can attempt to compute the probability measure of a set of paths using numerical techniques, but this is infeasible for systems with complex dynamics (e.g. generalized semi-Markov processes) or large state spaces. Existing CSL model checkers—ETMCC [3] and PRISM [5]—are limited to finite-state Markov chains.

To handle the generality of stochastic discrete event systems, Ymer implements the statistical model checking techniques, based on discrete event simulation and acceptance sampling, for CSL model checking developed by Younes and Simmons [12] (see also [10–Chap. 5]). To verify a formula $\mathcal{P}_{\geq\theta}[\varphi]$, Ymer uses discrete event simulation to generate sample execution paths and verifies the path formula $\varphi$ over each execution path. The verification result over a sample execution path is the outcome of a chance experiment (Bernoulli trial), which is used as an observation for an acceptance sampling procedure. Ymer implements both sampling with a fixed number of observations and sequential acceptance sampling. Ymer includes support for distributed acceptance sampling, i.e. the use of multiple machines to generate observations, which can result in significant speedup as each observation can be generated independently.

Ymer currently supports time-homogeneous generalized semi-Markov processes specified using an extension of the PRISM input language. PRISM and

* Supported by the Army Research Office (ARO) under contract no. DAAD190110485 and a grant from the Royal Swedish Academy of Engineering Sciences (IVA).

K. Etessami and S.K. Rajamani (Eds.): CAV 2005, LNCS 3576, pp. 429–433, 2005.

ETMCC work only with Markov processes, but support a richer set of properties than Ymer, including steady-state properties and unbounded until operators in path formulae. Ymer can use numerical techniques for continuous-time Markov chains (CTMCs) as it includes the hybrid engine from the PRISM tool for CTMC model checking. Numerical and statistical techniques can be used in combination to solve nested CSL queries for CTMCs as described by Younes et al. [11].

## 2 Distributed Acceptance Sampling

Statistical solution methods that use samples of independent observations are trivially parallelizable. One can use multiple computers to generate the observations, as noted already by Metropolis and Ulam [7–p. 340], and expect a speedup linear in the added computing power. To ensure that observations are independent, some care needs to be taken when generating pseudorandom numbers on each machine. Ymer uses the scheme proposed by Matsumoto and Nishimura [6], which encodes a process identifier into the pseudorandom number generator. This effectively creates a different generator for each unique identifier.

Ymer adopts a master/slave architecture (Fig. 1) for the distributed verification task. One or more slave processes register their ability to generate observations with a single master process. The master process collects observations from the slave processes and performs an acceptance sampling procedure. Each slave process is assigned a unique identifier by the master process to ensure that the slave processes use different pseudorandom number generators. The right side of Fig. 1 illustrates a typical communication session.

When using distributed sampling with a sequential test, such as Wald's [9] sequential probability ratio test, it is important not to introduce a bias against observations that take a longer time to generate. For probabilistic model checking, each observation involves the generation of a path prefix through discrete event simulation and the verification of a path formula over the generated path prefix. If we simply use observations as they become available, then the guarantees of the acceptance sampling test may no longer hold. For example, negative

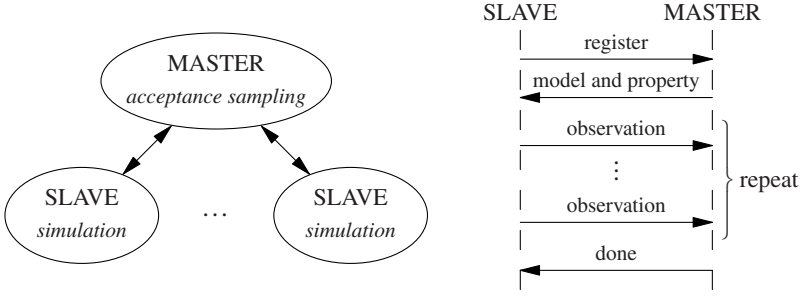


**Fig. 1.** Master/slave architecture and communication protocol for distributed acceptance sampling

observations for the path formula $\top\, \mathcal{U}^{[0,1000]}\, \Psi$ require simulation for 1000 time units, while positive observations may be fast to generate if $\Psi$ is satisfied early (cf. [10–Example 5.4]).

Such bias is avoided by committing, *a priori*, to the order in which observations are taken into account. Observations that are received out-of-order are buffered until it is time to process them. Ymer maintains a dynamic schedule of the order in which observations are processed. At the beginning, we schedule to receive one observation from each slave process in a specific order. We then reschedule the processing of the next observation for a slave process at the arrival of an observation. This gives us a schedule that automatically adjusts to variations in performance of slave processes without the need for explicit communication of performance characteristics.

To show the effect of distributed sampling, we use the model of an $n$-station symmetric polling system described by Ibe and Trivedi [4]. In this model, each station has a single-message buffer and the stations are attended by a single server in cyclic order. The server begins by polling station 1. If there is a message in the buffer of station 1, the server starts serving that station. Once station $i$ has been served, or if there is no message at station $i$ when it is polled, the server starts polling station $i + 1$ (or 1 if $i = n$). We verify the CSL property $m_1{=}1 \rightarrow \mathcal{P}_{\geq 0.5}[\top\, \mathcal{U}^{[0,20]}\, \mathit{poll}_1]$, which states that if station 1 is full, then it is polled within 20 time units with probability at least 0.5. We do so in the state where station 1 has just been polled and the buffers of all stations are full.

Fig. 2 shows the reduction in verification time for the symmetric polling system when using two machines to generate observations. The first machine is equipped with a 733 MHz Pentium III processor. If we also generate observations, in parallel, on a machine with a 500 MHz Pentium III processor, we get the relative performance indicated by the solid curve. The verification time with two machines is roughly 70 % of the verification time with a single machine. Fig. 3 shows the fraction of observations used from each machine, with $\mathrm{m}_1$ being

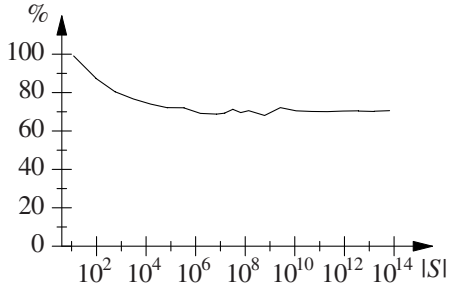


**Fig. 2.** Fraction of verification time as a function of state space size for the symmetric polling system when using two machines instead of one

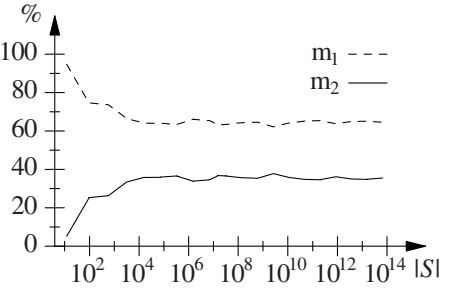


**Fig. 3.** Distribution of workload as a function of state space size for the symmetric polling system when using two machines to generate observations

the faster of the two machines. We can see that these fractions are in line with the relative performance of the machines.

## 3 Nested Probabilistic Operators

To illustrate the use of nested probabilistic operators, we consider the robot grid world described by Younes et al. [11]. A robot is moving in an $n \times n$ grid from the bottom left corner to the top right corner. The objective is for the robot to reach the top right corner within 100 time units with probability at least 0.9, while maintaining at least a 0.5 probability of periodically (every 9 time units) communicating with a base station. Let $c$ be a Boolean state variable that is true when the robot is communicating, and let $x$ and $y$ be two integer-valued state variables holding the current location of the robot. The CSL property $\mathcal{P}_{\geq 0.9}\left[\mathcal{P}_{\geq 0.5}[\top\ \mathcal{U}^{[0,9]}\ c]\ \mathcal{U}^{[0,100]}\ x{=}n \wedge y{=}n\right]$ expresses the desired objective.

The path formula for the outer probabilistic statement contains a probabilistic operator and cannot be verified without error with statistical techniques. Younes et al. [11] present a mixed solution method using statistical sampling for top-level probabilistic operators and numerical methods for nested probabilistic operators. Younes [10–Sect. 5.2] provides a purely statistical approach, which is made practical through the use of heuristics for selecting observation errors and *memoization* [8] to avoid repeated effort. Ymer implements both techniques.

Fig. 4 plots the verification time for the robot grid world property as a function of state space size. The results were generated on a machine with a 3 GHz Pentium 4 processor. The purely statistical approach is slower for smaller state spaces, but handles larger state spaces than the other two solution methods without exhausting memory resources (800 MB in this case). The dotted line shows where the property goes from being true to being false as the state space grows larger. The use of sequential acceptance sampling gives the peak in the curve for the mixed solution method, but the peak is not present in the curve for the purely statistical method thanks to memoization. The data shows that

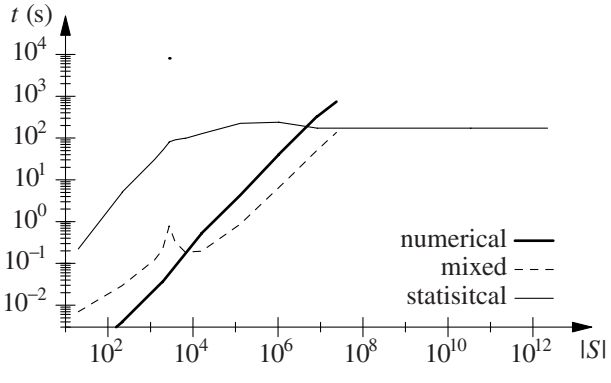


**Fig. 4.** Comparison of solution methods for robot grid world property with nested probabilistic operators

no method strictly dominates any other method in terms of verification time, although one should keep in mind that the correctness guarantees are different for all three methods. Numerical methods can guarantee high numerical accuracy, while statistical methods provide only probabilistic correctness guarantees.

## 4 Implementation Details and Availability

Ymer is implemented in C (the random number generator) and C++, and uses the CUDD package for symbolic data structures (BDDs and MTBDDs) used by the hybrid CTMC model checking engine. The part of the code implementing numerical analysis of CTMCs has been adopted from the PRISM tool. Ymer is free software distributed under the GNU General Public License (GPL), and is available for download at http://www.cs.cmu.edu/~lorens/ymer.html.

## References

1. Baier, C., Haverkort, B. R., Hermanns, H., and Katoen, J.-P. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, 2003.
2. Hansson, H. and Jonsson, B. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
3. Hermanns, H., Katoen, J.-P., Meyer-Kayser, J., and Siegle, M. A Markov chain model checker. In *Proc. 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1785 of *LNCS*, pages 347–362. Springer, 2000.
4. Ibe, O. C. and Trivedi, K. S. Stochastic Petri net models of polling systems. *IEEE Journal on Selected Areas in Communications*, 8(9):1649–1657, 1990.
5. Kwiatkowska, M., Norman, G., and Parker, D. Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal on Software Tools for Technology Transfer*, 6(2):128–142, 2004.
6. Matsumoto, M. and Nishimura, T. Dynamic creation of pseudorandom number generators. In *Monte-Carlo and Quasi-Monte Carlo Methods 1998*, pages 56–69. Springer, 2000.
7. Metropolis, N. and Ulam, S. M. The Monte Carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.
8. Michie, D. "Memo" functions and machine learning. *Nature*, 218(5136):19–22, 1968.
9. Wald, A. Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics*, 16(2):117–186, 1945.
10. Younes, H. L. S. *Verification and Planning for Stochastic Processes with Asynchronous Events*. PhD thesis, Computer Science Department, Carnegie Mellon University, 2005. CMU-CS-05-105.
11. Younes, H. L. S., Kwiatkowska, M., Norman, G., and Parker, D. Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer*, 2005. Forthcoming.
12. Younes, H. L. S. and Simmons, R. G. Probabilistic verification of discrete event systems using acceptance sampling. In *Proc. 14th International Conference on Computer Aided Verification*, volume 2404 of *LNCS*, pages 223–235. Springer, 2002.