Probabilistic Model Checking: A Comparison of GPMC versus PRISM

Mostafa Nouri Baygi[1], Ali Movaghar Rahimabadi[2]

[1] Department of Computer Engineering,
Sharif University of Technology, Tehran, Iran
nourybay@ce.sharif.edu

[2] Department of Computer Engineering, Faculty of Computer Engineering,
Sharif University of Technology, Tehran, Iran
movaghar@ce.sharif.edu

Abstract

We introduce a new tool for probabilistic model checking, GPMC, with a graphical user interface, and compare it with existing well-known tool in this scope PRISM. Some case studies are presented to show the efficiency and performance of it against PRISM. Two of these case studies are collected from several examples that were used for testing PRISM and another one is an extended simple example of a DTMC. We explain in each case study the powers and weaknesses of these tools, and find new methods for solving weaknesses.

Keywords

Probabilistic Model Checking, Probabilistic Models, Markov Chains, PRISM, DTMC, CTMC.

1. Introduction

Model checking is the most successful approach that's emerged for verifying requirements. A model-checking tool accepts system requirements or design (called models) and a property (called specification) that the final system is expected to satisfy. The tool then outputs yes if the given model satisfies given specifications and generates a counterexample otherwise.

In this paper we give an overview of probabilistic model checking, and two probabilistic model checkers, PRISM and GPMC. The first one is implemented in Birmingham University by Parker, Kwiatkowska and Norman [6], and the latter is a new tool implemented in Sharif University of Technology by Nouri under supervision of Prof. Movaghar. Then we compare these two tools with some simple case studies, and show the strengths of the new tool and its weaknesses.

2. Probabilistic Model Checking

In this section we introduce two probabilistic models, all variants of Markov chains. These two models are discrete time Markov chains (which feature probabilistic choice only), and continuous time Markov chains (which model continuous time and probabilistic choices).

2.1. Discrete Time Markov Chains

A DTMC is a Markov chain that uses discrete time for transitions. In a DTMC the frequency of a probabilistic edge being taken is determined by the distribution. Note that there is no notion of real time, though reasoning about discrete time is possible through state variables keeping track of time and 'counting' transition steps.

2.2. PCTL Logic

The logic PCTL (Probabilistic CTL) [4] replaces the existential and universal quantification of CTL with the *probabilistic operator* $\mathcal{P}_{\sim p}(.)$ where $p \in [0,1]$ is a *probability bound* or *threshold*, and $\sim \in \{\leq, <, \geq, >\}$. The syntax of state formulas ϕ of PCTL is:

$$\phi ::= true \mid a \mid \phi \wedge \phi \mid \neg\phi \mid \mathcal{P}_{\sim p}(\psi)$$

Where ψ is a path formula as:

$$\psi ::= X\phi \mid \phi\,\mathcal{U}^{\leq k}\phi \mid \phi\,\mathcal{U}\phi$$

A property of a model will always be expressed as a state formula. Path formulas only occur as the parameter of the probabilistic path operator $\mathcal{P}_{\sim p}(.)$. Intuitively, a state s satisfies $\mathcal{P}_{\sim p}(.)$ if the probability of taking a path from s satisfying ψ is in the interval specified by $\sim p$.

2.3. Continuous Time Markov Chains

Discrete time Markov chains can model *discrete time* only. Continuous time Markov chains have (finitely many) states that are discrete, a time parameter that ranges over $\mathbb{R}_{\geq 0}$, but do not allow non-determinism. Every transition is subject to an exponentially distributed random delay, and a *race condition* is used to deal with simultaneously enabled transitions.

2.4. CSL Logic

The logic CSL (Continuous Stochastic Logic) was introduced in [2] and extended in [3]. It is similar to the logic PCTL, but is designed to specify properties of CTMCs. CSL provides a way to describe steady-state and transient behaviors which are both elements of traditional CTMC analysis. It also allows specification of more involved properties using the probabilistic path operator of PCTL. The syntax is:

$$\phi ::= true \mid a \mid \phi \wedge \phi \mid \neg\phi \mid \mathcal{P}_{\sim p}(\psi) \mid \mathcal{S}_{\sim p}(\phi)$$

$$\psi ::= X\phi \mid \phi\,\mathcal{U}^{\leq t}\phi \mid \phi\,\mathcal{U}\,\phi$$

Where a is an atomic proposition, $\sim \in \{\leq, <, \geq, >\}$, $p \in [0,1]$ and $t \in \mathbb{R}_{\geq 0}$.

As for PCTL, $\mathcal{P}_{\sim p}(\psi)$ indicates that the probability of the path formula being satisfied from a given state satisfies the bound $\sim p$. Path formulas are the same for CSL as for PCTL except that the parameter t of the bounded until operator $\phi_1\,\mathcal{U}^{\leq t}\phi_2$ is a non-negative real rather than a non-negative integer. The $\mathcal{S}$ operator describes the steady-state behavior of the CTMC. The formula $\mathcal{S}_{\sim p}(\phi)$ asserts that the steady-state probability of being in a state satisfying ϕ meets the bound $\sim p$.

3. Tools

In this section we give a short description of PRISM and GPMC tools. These are two probabilistic model checkers that are compared in this paper.

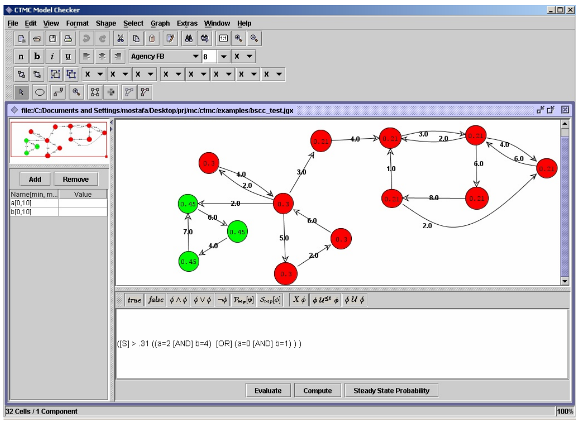
3.1. The PRISM Tool

PRISM [6] is a probabilistic symbolic model checker implemented using the CUDD package [8] to obtain BDD/MTBDD-based representation of probabilistic models. PRISM directly supports the DTMC, MDP and CTMC models and the specification languages PCTL and CSL. PRISM is available for download from [7].

3.2. The GPMC Tool

GPMC is a fully object oriented program written in Java for probabilistic model checking. It can model check DTMC and CTMC models. GPMC has a Graphical User Interface for inputting model. Users can draw the graph of model and its parameters, input their specifications and then verify model against these specification. GPMC consists of two main parts, each for one type of model. For graphical part of this tool we use JGraph [1] as a library for creating graphs of models. GPMC uses a graphical view of model, and is a new tool in the domain of probabilistic model checkers.
In this tool we move our attention from memory limitations and state space explosion and focus on efficiency of our tool and its running time for model checking. So in many situations its running time can be compared with tools such as PRISM that are written in low level languages.

GPMC, as stated, is fully object oriented. It is important because using this tool for programmers is a simple task. Although the graphical user interface is a good way for inputting models, our goal was not to implement only such a program. If we restricted our tool to model graphical inputs, model checking would be



```
([S] > .31 ((a=2 [AND] b=4)  [OR] (a=0 [AND] b=1) ) )
```

Fig. 1. View of GPMC while model checking a system.

simple, because only small models can be fed in this way, and the verifications of small models are easy because state space explosion and efficiency are not important in these cases. Therefore we use graphical inputs as the first way of reading models. We define a simple object oriented API for programmers, which can be used to define models of arbitrary complication and then model check them. We use this method for creating some case studies and comparing their results with results of PRISM. Fig. 1 shows a view of GPMC when it model checks a DTMC Model.

The methods we used for storing states and transitions are some variations of explicit methods and sparse matrices. These methods were used for best performance and efficiency and as it will be shown in the next section, their performances are comparable with PRISM.

4. Case Studies

In this section we present three case studies, gathered for comparing these two probabilistic model checkers. These results are obtained in a PC with Intel Celeron 2.4 GHz CPU with 256 MB of memory.

4.1. Dice Simulation

This case study considers probabilistic model, due to Knuth [5], which model fair dice using only fair coins. We use program written for PRISM as comparison to our tool. The detailed description of this model can be found in [7]. For this model we verify two state formulas. One of them can be stated as $\mathcal{P}_{\sim ?}(true\,\mathcal{U}(s = 7 \wedge d = k))$ for $k = 1..6$ and the other formula is $\mathcal{P}_{\sim ?}(true\,\mathcal{U}^{\leq k}(s = 7 \wedge d = 1))$ for different values of $k = 1..100$.

For more accuracy we put the model checking part in a loop with 10000 iterations and then divide result time by this number. For PRISM, also, we change the code to have more precision. In **Table 1** we can see the results of model checking.

4.2. The Cat Chases the Mouse

In this case study we focus on a model of chasing a mouse by a cat. In this model, there are some rooms, a

Table 1. Results for verfications of two state formulas with PRISM and GPMC.

State Formula	PRISM(ms)	GPMC(ms)
$\mathcal{P}_{=?}(true\,\mathcal{U}(s = 7 \wedge d = k)$ Total time for all k 's	3.928	0.1969
$\mathcal{P}_{=?}(true\,\mathcal{U}^{\leq 0}(s = 7 \wedge d = 0))$	0.0719	0.0047
	0.3031	0.0078
	0.2953	0.0125
to	0.3000	0.0141
	0.2984	0.0172
	0.3000	0.0203
$\mathcal{P}_{=?}(true\,\mathcal{U}^{\leq 9}(s = 7 \wedge d = 0))$	0.3016	0.0234
	0.3015	0.0266
	0.3016	0.0297
	0.3047	0.0328

mouse and a cat. In each step the mouse and the cat may change their rooms. Fig. 2 shows this model for 3 rooms. In this figure probability of moving cat to another room is 0.4 and remaining in its room is 0.2 and for mouse these probabilities are 0.3 and 0.4 respectively.

In Fig. 3 we can see the actual model of Fig. 2. as we can see the model consists of 9 states and 81 transitions. So if the number of rooms grows, the size of model increases very quickly.

We must find in this model the probability of cat reaching mouse if initially they were in different rooms. In Fig. 4 we see the results of model checking $\mathcal{P}_{=?}(true\,\mathcal{U}^{\leq k}(c = m))$ for different $k = 0..500$ and 30 rooms.

As it can be seen in this chart, the running times of model checking this formula are equal for two model checkers, till $k = 350$. And for this bound and greater,

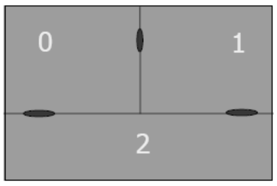


Fig. 2. Three rooms that a mouse and a cat moves between them.

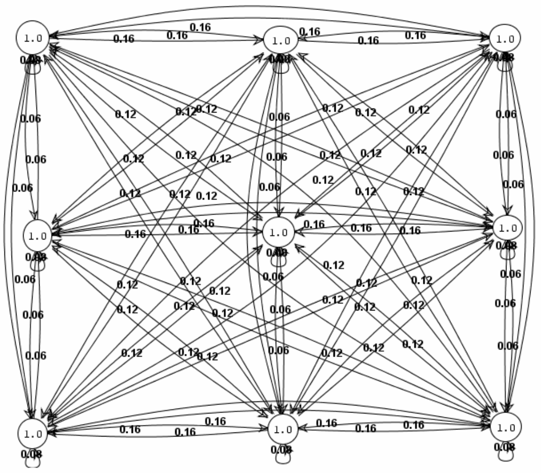


Fig. 3. Model of moving mouse and cat between rooms.

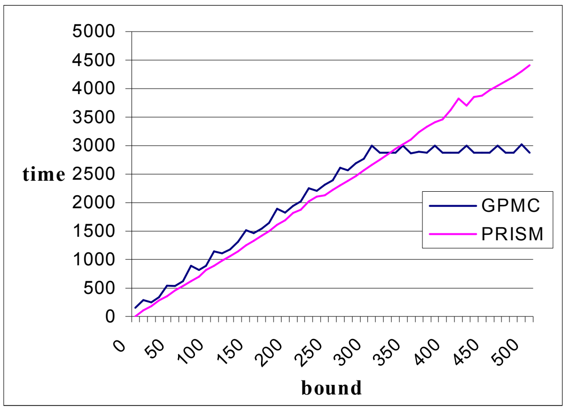


Fig. 4. Running time of model checking bounded until for model "cat and mouse" in PRISM and GPMC.

the running time for GPMC remains constant. This is because this tool sees no change in the probability as go further so concludes we reach the steady states probabilities and stop going further.

4.3. Molecular Reactions

This case study, as the first one, is obtained from case studies used for PRISM, so we use the program written for PRISM to compare it with results of GPMC.

In this case study we consider a molecular reaction taken from Ehud Shapiro's lecture notes on Biomolecular Processes as Concurrent Computation. More details about this case study can be read in [7].

Note that, for chemical reaction we consider, the rate of the reaction is the base rate multiplied by the product of the number of molecules of each type that take part in the reaction. The reaction that studied was $Na + Cl \leftrightarrow Na^{+} + Cl^{-}$. The reaction starts with an equal number of Na and Cl. The rate of forward reaction is

$$100.Na.Cl = 100.Na^{2}$$

And the rate of backward reaction is

$$10.Na^{+}.Cl^{-} = 10.(N - Na)^{2}\,.$$

We want to know the probability that in steady state there is some Na molecule, formally $S_{=?}(Na > 0)$. We can see a visual view of this model for initial number of Na, $N = 10$ in Fig. **5**.

The running time of model checking of this model for different number of N is plotted in Fig. 6. As it can be seen the running time for GPMC grows quickly. Maybe it is because we use LU decomposition for solving linear equation systems whereas in PRISM it uses different ways of iterative methods for solving them.

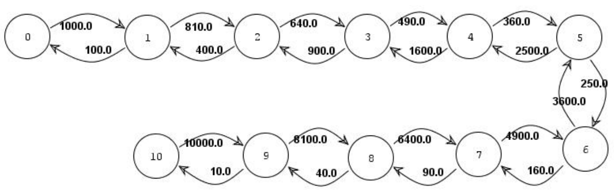


Fig. 5. Model of molecule reactions for $N = 10$.

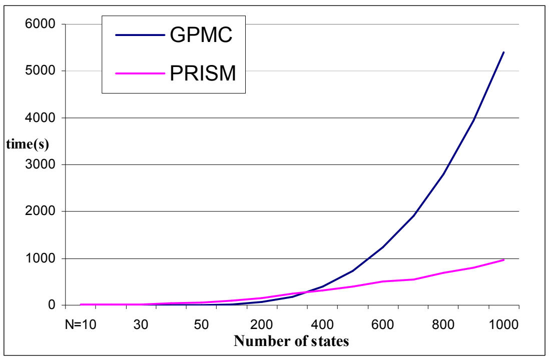


Fig. 6. The running time of model checking steady state formula in PRISM and GPMC.

In doing this case study we find a bug in PRISM. As stated before, there are different methods for solving linear equation systems. The default one in this tool solves the equations of this case study and computes the probabilities wrongly.

We model check this formula for different value of N and t. The results of this model checking can be viewed in Fig. 7. For PRISM the running time increases as number of states or time bound increases. In GPMC it is so, but in some places, suddenly it fallen and then increase very slowly. It is because in GPMC, when computation reaches its ceiling, it assumes that the probability is near the unbounded until probability, as time bound is so great. Therefore it computes unbounded until probabilities which are much faster than time bounded one. Regarding this situation, reaching its ceiling capability, we ran a test on both GPMC and PRISM and found another bug in PRISM. The formula for this test is $\mathcal{P}_{\sim p}[true\,\mathcal{U}^{\leq t}(Na = N)]$.

In our initial state $Na = N$, therefore the probability for this formula is 1 without any concern to t. But the results of model checking this formula in GPMC and PRISM differ in some cases (Table 1). We see in this table that for bound 22 and greater the probability that PRISM computes is wrong, but the time used for computing does not change significantly.

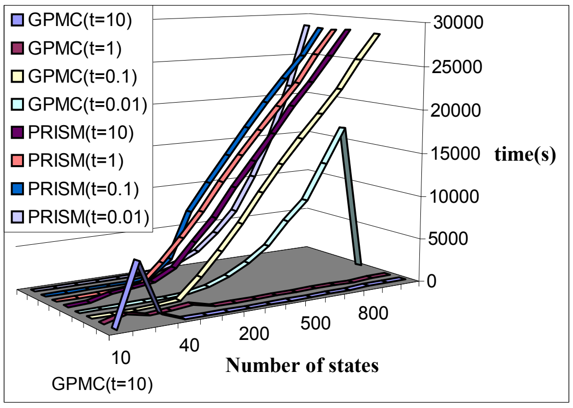


Fig. 7. The running time of model checking $\mathcal{P}_{\sim p}\ [\ true\ \mathcal{U}^{\leq t}\ (Na = 0)\ ]$ for different number of states and different time bounds.

Table 2. A comparison of the two tools that shows a bug in PRISM.

$N = 1000$	PRISM		GPMC	
	Time(ms)	Prob.	Time(ms)	Prob.
$\mathcal{P}_{\sim p}\ [\ true\ \mathcal{U}^{\leq 0.2}\ (Na = 1000)\ ]$	250	1	270	1
$\mathcal{P}_{\sim p}\ [\ true\ \mathcal{U}^{\leq 20}\ (Na = 1000)\ ]$	260	1	256	1
$\mathcal{P}_{\sim p}\ [\ true\ \mathcal{U}^{\leq 22}\ (Na = 1000)\ ]$	420	0	257	1
$\mathcal{P}_{\sim p}\ [\ true\ \mathcal{U}^{\leq 200}\ (Na = 1000)\ ]$	840	0	254	1

5. Conclusion And Future Works

In this paper we implement a probabilistic model checking tool, GPMC, and use it for some case studies. So far we have concentrated our efforts on efficient implementation of the techniques used previously in PRISM, a Probabilistic Symbolic Model Checker.

One of the main motivations for our work in this paper was the weaknesses of existing implementations of probabilistic model checkers. In this tool we focus on improving performance and running time of verification. Some case studies are presented and their results are delivered. These results show that the model checking can be done very faster in comparison to the old tools.

In the list of future works we put improving its performance, especially in places that it shows bad results against PRISM, such as steady state of CTMC. By extending case studies for this tool, we can find bugs and problems, and discover its benefits over older tools.

We should also implement some interfaces for other tools, which create some input for this tool, CTMC or DTMC. By doing this we can extend the number of case studies for this tool very rapidly.

References

[1] G. Alder, *Design and implementation of the JGraph Swing component*, http://www.jgraph.com/paper.html.

[2] A. Aziz, K. Sanwal, V. Singhal, R. K. Brayton, *Verifying continuous time Markov chains*, In Proc. 8th International Conference on Computer Aided Verification (CAV'96), 269–276, 1996.

[3] C. Baier, J. P. Katoen, H. Hermanns, *Approximate symbolic model checking of continuous-time Markov chains*. In Proc. 10th International Conference on Concurrency Theory (CONCUR'99), 146–161, 1999.

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

[5] D. E. Knuth, A. C. Yao, *The complexity of non-uniform random number generation*. In Algorithms and Complexity: New Directions and Recent Results, Academic Press, New York, 1976

[6] M. Kwiatkowska, G. Norman, and D. Parker, *PRISM: Probabilistic symbolic model checker*. In Proc. TOOLS'02, 200–204, 2002.

[7] PRISM web site, http://www.cs.bham.ac.uk/˜dxp/prism.

[8] F. Somenzi, CUDD: *Colorado University decision diagram package*. Public software, Colorado Univeristy, Boulder, http://vlsi.colorado.edu/˜fabio/, 1997.