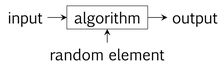
THE WORLD'S LARGEST BENCHMARK SUITE FOR PROBABILISTIC MODEL CHECKING

Yash Dhamija, Syyeda Zainab Fatmi, Maeve Wildes, Franck van Breugel

BACKGROUND

A randomized algorithm is an algorithm that makes random choices during execution, so the behaviour can vary even on a fixed input.



Randomized algorithms are pervasive in the hottest fields including computer games, AI, and cryptography. Randomized algorithms:

- allow previously unsolvable problems to be solved
- can be drastically more efficient than deterministic algorithms
- are less vulnerable to adversaries

"It may seem unlikely that such a consultation with a "throw of the dice" could speed up a computation. […] It turns out that in certain cases this approach effects dramatic improvements."

- MICHAEL OSER RABIN

INTRODUCTION

Software bugs cost the worldwide economy trillions of dollars per year.

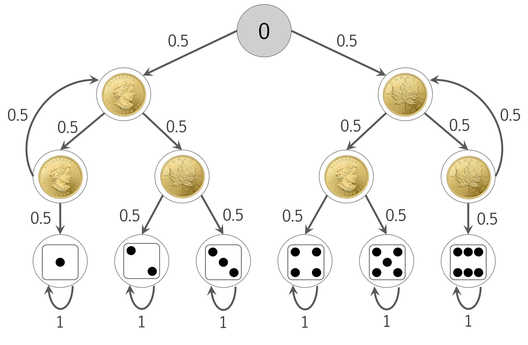
Software testing is most commonly used to find bugs in software. However, running a test on randomized software does not guarantee that all possible executions are checked.

"Program testing can be used to show the presence of bugs, but never to show their absence!"

- EDSGER WYBE DIJKSTRA

Probabilistic model checking is a formal technique that can be used to detect bugs in the presence of randomness. When using this technique:

- the software system is modelled as a state machine
- the specifications of the software system are expressed as logic formulas



The model checker either confirms that the property holds in the model or provides a counterexample.

MOTIVATION

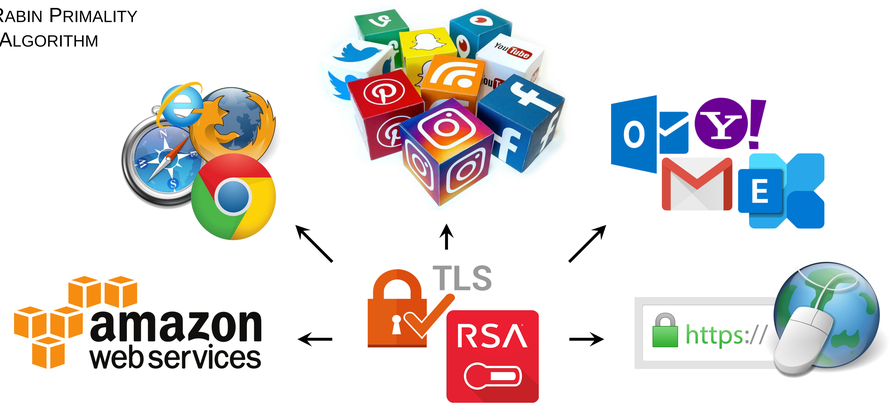
The field of probabilistic model checking is lacking good benchmarks to evaluate new tools and techniques. Currently, researchers

- either consider less than a handful of realistic probabilistic models, which provides little confidence in the results
- or use randomly generated probabilistic models, which is not useful as they tend to not have the same characteristics as models encountered in practice

Labelled Markov chains are the most abstract and most used type of probabilistic models. The main goal of this research is to generate a large collection of realistic instances of Markov chains, which can be used as a benchmark suite to evaluate techniques and tools that support probabilistic model checking.

METHODOLOGY

MILLER-RABIN PRIMALITY TESTING ALGORITHM

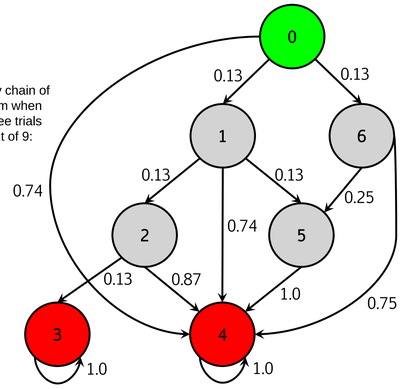



```
// 2. Choose a uniformly at random from Zn\{0}
Random random = new Random();
int a = random.nextInt(n - 1) + 1;

// 3. For i = 0 to r compute bi = a^(2^i * R) mod n
double[] b = new double[r + 1];
for (int i = 0; i <= r; i++) {
    b[i] = mod(a, (int) (Math.pow(2, i) * R), n);
}

// 4. If a^(n-1) = br not congruent to 1 (mod n) then return COMPOSITE
// 5. If a^(R) = b0 congruent to 1 (mod n) then return PRIME
if (b[r] != 1) {
    return false;
} else if (b[0] == 1) {
    return true;
}
```

The Markov chain of the algorithm when run with three trials and an input of 9:



1. Implementing new models

The following steps were taken to develop probabilistic models:

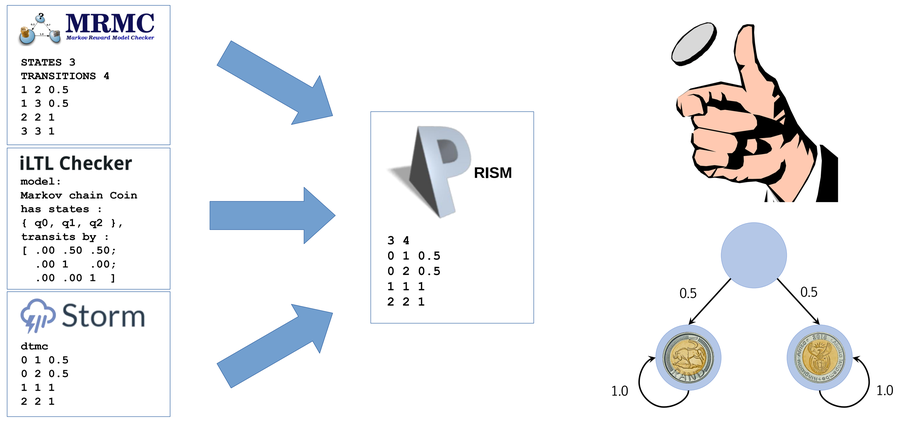
1) Implement randomized algorithms in Java
2) Convert Java code to Java PathFinder (JPF)-compatible Java code
3) Extract the embedded Markov chain using JPF
   - JPF keeps track of the random choices and creates a probabilistic model in which all possible outcomes are considered.

Randomized algorithms are prevalent in some of the hottest areas in computer science. One of these algorithms, the Miller-Rabin primality test (above) is used in RSA encryption which has many important cryptographic applications.
The algorithms that we implemented were extremely varied in their use, including:

- Graph algorithms such as min-cut, max-cut, and all-pairs-shortest-path
- Data structures such as treaps, skip lists and universal hash maps
- Number theory algorithms such as quadratic residues, polynomial root, and primality testing

2. Collecting and converting existing models

We collected all existing models from various model checking tools. We implemented 3 converters that take as input a model in the format of MRMC, iLTLChecker, or Storm and output a model in the format of PRISM, the most commonly used model checker.



We implemented a converter that takes as input a continuous time Markov chain (CTMC) and produces as output a discrete time Markov chain (DTMC). CTMCs have a rate associated with each transition, whereas DTMCs have a probability associated with each transition. To go from CTMC to DTMC, the probability $P(s,s')$ that each transition will occur is abstracted from the transition rate $R(s,s')$ using the following formula:

$$P(s,s') = \begin{cases} \dfrac{R(s,s')}{E(s)} \; if \; E(s) > 0 \\ 1 \; if \; E(s) = 0 \; and \; s = s' \\ 0 \; if \; E(s) = 0 \; and \; s \neq s' \end{cases}$$

where

$$E(s) = \sum_{s' \in S} R(s,s')$$

PRISM CTMC

```
27 52
0 1 70
1 0 1
1 2 70
2 1 2
2 3 70
3 2 3
3 4 70
4 3 4
4 5 70
5 4 5
5 6 70
```

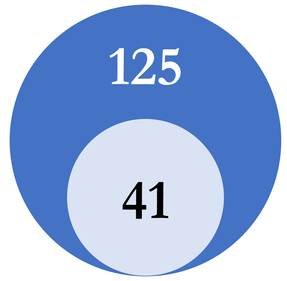
PRISM DTMC

```
27 52
0  1  1.0000
1  0  0.0141
1  2  0.9859
2  1  0.0278
2  3  0.9722
3  2  0.0411
3  4  0.9589
4  3  0.0541
4  5  0.9459
5  4  0.0667
5  6  0.9333
```

RESULTS

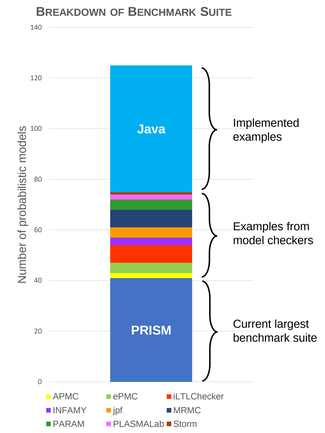
As previously discussed, we created the benchmark suite by gathering and translating numerous models from existing probabilistic model checkers and by developing realistic models in Java.

The benchmark suite contains 125 models, making it thrice as big as any current individual collection.



Furthermore, 85% of the models are parametric and can be instantiated in many ways.

BREAKDOWN OF BENCHMARK SUITE



We wrote Java applications to record certain properties of each example. Following are the properties of the Miller-Rabin Primality test models, instantiated with various different parameters:

Input	Trials	States	Transitions	% Zeros	% Ones	% Probabilistic	% Uniform
5	1	3	4	55.55	22.22	33.33	0
	2	5	7	72.00	12.00	40.00	0
	3	7	10	79.59	8.16	42.86	0
	4	9	13	83.95	6.17	44.44	0
	5	11	16	86.78	4.96	45.45	0
9	1	3	4	55.55	22.22	33.33	0
	2	5	8	68.00	12.00	40.00	0
	3	7	13	73.47	6.12	57.14	0
	4	9	17	79.01	4.94	55.55	0
	5	11	23	80.99	2.48	72.73	0
11	1	3	4	55.55	22.22	33.33	0
	2	5	7	72.00	12.00	40.00	0
	3	7	10	79.59	8.16	42.86	0
	4	9	13	83.95	6.17	44.44	0
	5	11	16	86.78	4.96	45.45	0

IMPACT

This collection will be made open-source, providing the research community with much-needed benchmarks to test probabilistic model checkers.

"When a field has good benchmarks, we settle debates and the field makes rapid progress."

- DAVID ANDREW PATTERSON

In the future, the models can be analyzed to:

- identify common properties
- better understand the properties that make these models of critical importance in the testing environment
- assess the impact of parameters on the properties of a model
- gain insight on how to choose meaningful parameters for randomized algorithms

REFERENCES

- Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A Storm is coming: A modern probabilistic model checker. In Rupak Majumdar and Viktor Kuncak, editors, *Proceedings of the 29th International Conference on Computer Aided Verification,* volume 10427 of *Lecture Notes in Computer Science,* pages 592–600, Heidelberg, Germany, July 2017. Springer-Verlag.
- Joost-Pieter Katoen, Ivan Zapreev, Ernst Moritz Hahn, Holger Hermanns, and David Jansen. The ins and outs of the probabilistic model checker MRMC. *Performance Evaluation,* 68(2):90–104, February 2011.
- Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Proceedings of the 23rd International Conference on Computer Aided Verification,* volume 6806 of *Lecture Notes in Computer Science,* pages 585–591, Snowbird, UT, USA, July 2011. Springer-Verlag.
- YoungMin Kwon and Gul Agha. Linear inequality LTL (iLTL): A model checker for discrete time Markov chains. In Jim Davies, Wolfram Schulte, and Mike Barnett, editors, *Proceedings of the 6th International Conference on Formal Methods and Software Engineering,* volume 3308 of *Lecture Notes in Computer Science,* pages 194–208, Seattle, WA, USA, November 2004. Springer-Verlag.
- Gary L. Miller. Riemann's hypothesis and tests for primality. In William C. Rounds, Nancy Martin, Jack W. Carlyle, and Michael A. Harrison, editors, *Proceedings of the 7th Annual ACM Symposium on Theory of Computing,* pages 234–239, Albuquerque, New Mexico, USA, May 1975. ACM.
- Naval History and Heritage Command. The First "Computer Bug", 1947. [Online; accessed August 12, 2019].
- Michael Oser Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory,* 12(1):128–138, February 1980.
- Tricentis. Software Fail Watch, 2017. [Online, accessed August 12, 2019].
- Willem Visser, Klaus Havelund, Guillaume Brat, SeungJoon Park, and Flavio Lerda. Model checking programs. *Automated Software Engineering,* 10(2):203–232, April 2003.
- Xin Zhang and Franck van Breugel. Model checking randomized algorithms with Java PathFinder. In *Proceedings of the 7th International Conference on the Quantitative Evaluation of Systems,* pages 157–158, Williamsburg, VA, USA, September 2010. IEEE.