# Model Fuzzing: Testing Probabilistic Model Checkers via Mutation-based Fuzzing

September 16, 2024

## Problem description

Model checking is a widely used technique for verifying different types of software systems, such as spacecraft controllers, airline reservation systems, E-commerce protocols, communication systems for household appliances [1], etc. Nevertheless, for all these applications to provide reliable results, the model checkers themselves have to be correct. As formally verifying a verifier is usually not feasible for real-world, complex tools [3], more practical approaches rely on testing.

*The goal of this project is to design a testing technique based on mutational fuzzing to automatically identify errors in model checkers. In particular, we will focus on probabilistic model checkers, which are used for modeling and analyzing systems with probabilistic behaviors. Given a set of seed models (e.g., Markov chains) together with the properties they should fulfill (e.g., the expected probability of an event), we will define various mutations to modify the models, such that their effect on the property is known by construction.*

## Example

Let us consider the example from Figure 1, which illustrates a case study from the PRISM model checker[1]. It models a die using a fair coin (according to Knuth and Yao's algorithm [5]) and can be represented as a discrete time Markov chain (DTMC): the initial state is marked with 0, the red states represent the die's outcomes. The other states correspond to tossing a coin. If the result is heads (which for a fair coin has 50% chance), then the upper branch determines the next state, otherwise the lower branch determines it. PRISM [6] can prove the correctness of this algorithm by showing that the probability of reaching any red state (that is, of throwing 1, 2, ..., 6) is 1/6.

[1]https://www.prismmodelchecker.org/casestudies/dice.php

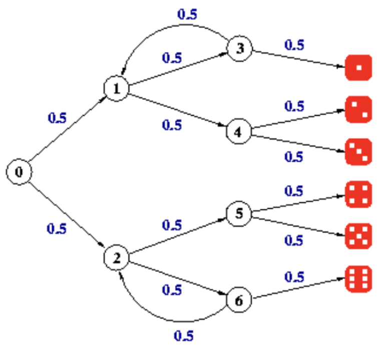

Figure 1: Modeling a die using a fair coin

## Approach

We will start from an existing set of benchmarks (e.g., the PRISM benchmarks suite [7]). They include various types of models (e.g., DTMCs, Markov decision processes (MDPs), continuous time Markov chains (CTMCs)) and their corresponding properties. Since these benchmarks are already used for testing model checkers, we can assume that the properties are correct. We will then define different categories of mutations that can be applied to the models, such as:

1. modifying the probabilities of some of the transitions (for DTMCs) or their rates (for CTMCs)
2. modifying the guards[2] for some of the transitions
3. inserting new transitions (and their corresponding probabilities/rates)
4. removing some of the transitions

and we will determine their effect on the checked property. We will also investigate how the mutations we defined can be composed.

In Figure 1, if we modify the probability of the transition $2 \rightarrow 5$ to 0.25 (and of the transition $2 \rightarrow 6$ to 0.75, such that the sum of the probabilities remains 1), then the likelihood of obtaining a 4 or a 5 should decrease, and the likelihood of obtaining a 6 should increase. The exact values of the updated probabilities may be hard to determine automatically, for arbitrary modifications but our algorithm should be able to identify that the faces 1, 2, and 3 are not affected by the modification (their probabilities remain unchanged).

To evaluate our approach, we will use the modified benchmarks for testing state-of-the-art probabilistic model checkers, such as PRISM and Storm [4]. If their source code is publicly available, we will also measure the impact of our mutations on the coverage.

[2]The guards are predicates expressing when a transition can take place. If not explicitly written, they are equivalent to *true* (as in Figure 1).

As possible extensions, we could define a transformation that converts a given model into a different representation. Storm, for examples, support various input formats[3] (PRISM, JANI, explicit transition system, etc.). The conversion between representations should not affect the result of the model checker under test. Moreover, we could also fuzz the space of available options, as proposed for SMT solvers in [8].

## Related Work

The problem of validating different types of program analysis tools (such as compilers, static analyzers, symbolic execution engines, SMT solvers) has received a lot of attention from the research community (see [2] for an overview). However, testing probabilistic model checkers is an area much less explored. To the best of our knowledge, the only work that targets model checkers (but not probabilistic ones) is [9]: the approach starts from deterministic C programs and leverages their concrete executions to automatically synthesize reachability properties. Our proposed technique is more generic, supports various types of models (CTMCs, DTMCs, MDPs, etc.) and arbitrary probabilistic computation tree logic (PCTL) properties.

## Prerequisites

The student is expected to have good programming skills and basic knowledge of probability. Prior experience with model checkers is a plus.

## Opportunities

The student will have the chance to gain a deep understanding of probabilistic model checkers and to learn about state-of-the-art fuzzing techniques.

## Contact

Alexandra Bugariu: bugariua@mpi-sws.org

[3]https://www.stormchecker.org/documentation/background/languages.html

## References

[1] Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

[2] Alexandra Bugariu. *Automatically Identifying Soundness and Completeness Errors in Program Analysis Tools.* PhD thesis, ETH Zürich, 2022.

[3] Cristian Cadar and Alastair Donaldson. Analysing the program analyser. ACM, 2016.

[4] Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. The probabilistic model checker storm. *Int. J. Softw. Tools Technol. Transf.*, 24(4):589–610, aug 2022.

[5] Donald Knuth and Andrew Yao. *Algorithms and Complexity: New Directions and Recent Results*, chapter The complexity of nonuniform random number generation. Academic Press, 1976.

[6] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[7] Marta Kwiatkowska, Gethin Norman, and David Parker. The PRISM benchmark suite. In *Proc. 9th International Conference on Quantitative Evaluation of SysTems (QEST'12)*, pages 203–204. IEEE CS Press, 2012.

[8] Peisen Yao, Heqing Huang, Wensheng Tang, Qingkai Shi, Rongxin Wu, and Charles Zhang. Fuzzing smt solvers via two-dimensional input space exploration. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2021, pages 322–335, New York, NY, USA, 2021. Association for Computing Machinery.

[9] Chengyu Zhang, Ting Su, Yichen Yan, Fuyuan Zhang, Geguang Pu, and Zhendong Su. Finding and understanding bugs in software model checkers. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, pages 763–773, New York, NY, USA, 2019. Association for Computing Machinery.