THE EFFICIENCY COMPARISON OF THE PRISM AND STORM PROBABILISTIC MODEL CHECKERS FOR ERROR PROPAGATION ANALYSIS TASKS

T. Fabarisov[1], N. Yusupova[1], K. Ding, A. Morozov[2], K. Janschek[2]
Faculty of Computer Science and Robotics, Ufa State Aviation Technical University Ufa, Russia[1]
Technische Universität Dresden Germany[2]
flatagir@gmail.com, yussupova@ugatu.ac.ru, kai.ding@tu-dresden.de, andrey.morozov@tu-dresden.de, klaus.janschek@tu-dresden.de

***Abstract**: Dual-graph Error Propagation Model (DEPM) is a stochastic framework developed in our lab that captures system properties relevant to error propagation processes such as control and data flow structures and reliability characteristics of single components.*
The DEPM helps to estimate the impact of a fault of a particular component on the overall system reliability. The probability of a system failure, Mean Time Between Failures and Mean Time to Repair, and the expected number of failures are the quantitative results of the DEPM-based analysis. In addition, the DEPM provides an access to the powerful techniques from Probabilistic Model Checking (PMC) that allow the evaluation of advanced, highly customizable, time-related reliability properties, e.g., "the probability of a system failure during certain time units after the occurrence of a certain combination of errors in certain system components".

For this reason, the DEPM models are automatically transformed to discrete-time Markov chain (DTMC) models and are analyzed using the state of the art probabilistic model checker PRISM. However, the new promising model checker Storm has been recently released.

This paper presents the results of the efficiency comparison of these two model checkers based on the automatically generated set of the DTMC models that are typical for the error propagation analysis tasks. Several computation engines that are supported by both model checkers have been taking into account. The paper also gives the general description of the DEPM workflow with the focus on the PMC interface.

***KEYWORDS**: ERROR PROPAGATION MODEL, RELIABILITY, SAFETY, DEPENDABILITY, MODEL-BASED SYSTEMS, MODEL-BASED ANALYSIS, PROBABILISTIC MODEL CHECKING, CONTROL FLOW, DATA FLOW, OPTIMIZATION.*

1. Introduction

Evaluation of the system reliability characteristics is an important part of the safety analysis which helps to assess the degree of dependability, performance, and safety assurance of the future system and helps to determine whether or not system (or system unit) will meet the qualitative requirements, e.g., "the expected mean time between failures for a given scenario of particular system part" or "the minimum time before the battery leakage given certain operating temperature".

Estimation of these characteristics allows safety engineers to identify hazards, analyze and assess safety risks prior to implementation of the system design. The safety assessment must be evaluated to ensure that safety requirements have been met. The general goal is to minimize the hazard-related risks in safety-critical industrial domains, where a failure or malfunction may result in environmental harm, severe equipment damage or even serious injuries of the people.

2. Probabilistic Model Checking and DEPM

Fault likelihood of the system components and overall system behavior in the context of error propagation can be expressed in formalized metrics – reliability properties.

These metrics can be divided into three categories [1]:

- quantitative reachability: the probability of reaching some state of the system model, e.g., "probability that more than 25 percent of outputs are erroneous";
- qualitative reachability: whether the probability of reaching some set of states of the system model is 0 or 1, e.g., "probability that a leader is eventually elected";
- expected reachability: expected accumulated reward or cost to reach some set of states of the system model, e.g., "expected number of steps until termination".

The Dual-graph Error Propagation Model (DEPM) is a mathematical abstraction [2, 5] of the system properties such as control and data flow structures and reliability characteristics of single components, which are vital for the determination of the error propagation processes. It is a useful analytical tool for assessing the impact of specific errors and errors on the overall behavior of the system. DEPM-based analysis is used to assess the reliability properties of a system in the context of error propagation. DEPM is represented by the set-based mathematical notation:

$$DEPM := \langle E, D, A_{CF}, A_{DF}, C \rangle$$

Where E is a non-empty set of elements; D is a set of data storages; A_{CF} is a set of directed control flow arcs, extended with control flow decision probabilities; A_{DF} is a set of directed data flow arcs; C is a set of conditions of the elements.

The OpenErrorPro [3, 4] is an analytical software developed in our lab that supports the system analysis with the DEPM. On the input there are baseline models that describe the target system. Several parsers transform the baseline models into the DEPM models [5]. A screenshot of the OpenErrorPro is shown in Figure 1.

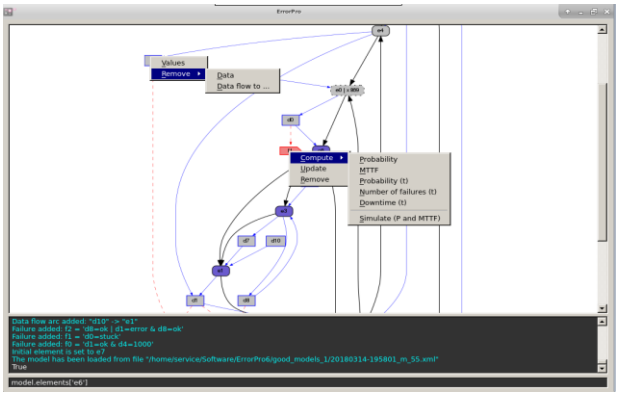

***Fig 1**. A screenshot of the OpenErrorPro*

Reliability properties can be used for evaluation of such system characteristics as Mean Time to Failure (MTTF), Mean Time Between Failure (MTBF), Failure in Time (FIT), Mean Time to Repair (MTTR), and etc [6]. These reliability metrics refer to the system itself or to a specific component. Despite the fact that MTBF remains the main metric of system reliability for most cases, it is often being considered as no more relevant and, in general, widely incomprehensible. However, it is still a useful metric in consideration of its restrictions. For that reason, the DEPM also

provides an access to the powerful techniques from the Probabilistic Model Checking (PMC) [7, 8].

These techniques allow the evaluation of advanced, highly customizable, time-related reliability properties, e.g., "*the probability of a system failure during certain time units after the occurrence of a certain combination of errors in certain system components*".

In order to calculate probabilistic properties DEPM models must be transformed to discrete-time Markov chain models (DTMC). In the DTMC models each state of the system has a probability distribution to it successor states. Whenever the system is in a certain state, a type of random experiment is conducted to determine the state to which the system moves at the next step. The number of steps that the system takes is counted by non-negative integers, therefore, in the DTMC models time is discrete. DTMC models are described in a PRISM language. PRISM model checker [9] is a powerful stochastic probabilistic model analyser. It uses its own notation to describe DTMC models and PRISM's property specification language to describe reliability properties of the system. A good example of PRISM model and properties is given in [10]. Generated DTMC models consist of the initial state; of the set of states to which model can move to; of the rewards: values associated with certain states of the model; of the labels: sets of states that are of particular interest. The reliability properties can be calculated for DTMC model.

3. Storm model checker

Storm [11] is a tool for the analysis of systems involving random or probabilistic phenomena. Given an input model and a quantitative specification, it can determine whether the input model conforms to the specification. It has been designed with performance and modularity in mind. The tool has been developed at the Chair for Software Modeling and Verification at RWTH Aachen University.

Storm is built around discrete- and continuous-time Markov models:

- Discrete-Time Markov Chains
- Markov Decision Processes
- Continuous-Time Markov Chains
- Markov Automata [12]

According to the author's website, Storm model checker has several characteristics that made it a promising tools, which has the efficient computation core, modularity of the modules, support of various input languages, and the python interface. Because the properties of the analyzing systems are very different Storm has several engines that determine which data structures are used to build and analyze the model accordingly. Supporting of the several input modeling languages helps to overcome the problem that occurs when the target model is already created by another tool and cannot be transferred to another toolset without translation [13].

In order to help a user to abstract from some of core code and focus on the computation algorithms, the developers of the Storm had published a Strompy python interface, which provides a python API to the main functionality of the Storm code. For the efficient benchmarking of Storm model checker, we used Stormpy – python API for Storm. It allows to check models and verify properties right in the python environment [14]. On the input Stormpy requires a path to the file with a symbolic description of the model in the PRISM language format and a string that contains properties in the prism-property format. An example of the code is shown in Listing 1. An example of the symbolic description of the simple model in the PRISM language format is shown in Listing 2.

```
path = "/path/to/model/"
prism_program = stormpy.parse_prism_program(path)
formula_str = "P=? [F s=2]"
properties = stormpy.parse_properties_for_prism_program
(formula_str, prism_program)
model = stormpy.build_model(prism_program, properties)
result = stormpy.model_checking(model, properties[0])
print(result)
```

Listing 1. An example of the Stormpy usage

```
dtmc
module die
s : [0..7] init 0;
d : [0..6] init 0;
[] s=0 -> 0.5 : (s'=1) + 0.5 : (s'=2);
[] s=1 -> 0.5 : (s'=3) + 0.5 : (s'=4);
[] s=2 -> 0.5 : (s'=5) + 0.5 : (s'=6);
[] s=3 -> 0.5 : (s'=1) + 0.5 : (s'=7) & (d'=1);
[] s=4 -> 0.5 : (s'=7) & (d'=2) + 0.5 : (s'=7) & (d'=3);
[] s=5 -> 0.5 : (s'=7) & (d'=4) + 0.5 : (s'=7) & (d'=5);
[] s=6 -> 0.5 : (s'=2) + 0.5 : (s'=7) & (d'=6);
[] s=7 -> (s'=7);
endmodule
rewards "coin_flips"
[] s<7 : 1;
endrewards
```

Listing 2. An example of the analyzed model

It has been tested which one of two model checkers is more efficient. In order to test time efficiency of Storm and PRISM model checkers, two scripts have been written. For the testing, 2464 PRISM models have been generated. For every model two properties have been checked. Every model has been checked four times. Therefore, for every model we got 8 checked properties. Mean time of property analysis per every model has been calculated. The snippet of results and the total result of the comparison are presented in the Table 1. The comparison has shown that Storm takes 0,96 seconds to analyze one property, PRISM analyzes one property in 2,64 seconds which is almost three times more than Storm. Despite the impressive results and promising novelty of Storm, there are several reasons that restrict us from implementing Storm into the ErroPro.

Storm is not capable of computing Steady-state and Transient probabilities. In PRISM if the model is a CTMC or DTMC, it is possible to compute corresponding vectors of steady-state or transient probabilities directly (rather than indirectly by analyzing a property which requires their computation) [15]. Because the Storm has many different input formats and it depends on the format how this distribution would have to be specified. PRISM has an advantage in the sense that it supports only one input format.

Table 1. *Results of the Storm and PRISM comparison (seconds)*

Model	*Mean_time STORM*	*Mean_time PRISM*
m_1209.xml_prism.pm	1,1129	2,5478
m_1208.xml_prism.pm	0,6677	2,6055
m_1207.xml_prism.pm	1,2605	3,2407
…	…	…
m_2.xml_prism.pm	1,2075	2,6420
m_55.xml_prism.pm	1,1163	2,8417
m_33.xml_prism.pm	0,7144	2,3773
Min	0,64	2,27
Max	1,98	7,78
Mean	0,96	2,64

Table 2. *Results of the different PRISM engines comparison (seconds)*

Model	*Mean time (explicit)*	*Mean time (MTBDD)*	*Mean time (sparse)*	*Mean time (hybrid)*
m_47.pm	2,4231	2,3996	2,4512	2,4499
m_32.pm	2,4276	2,4223	2,4540	2,3831
m_38. pm	2,4608	2,3630	2,3937	2,3855
…	…	…	…	…
m_1096.pm	2,4816	2,3926	2,4845	2,4879
m_1200.pm	2,3484	2,3605	2,4053	2,4523
m_1261.pm	2,7282	19,3512	2,5472	2,4892
Mix	2,3026	2.3230	2,3254	2,3595
Min	2,8577	108,3340	3,7128	3,6529
Mean	2,4819	5,7274	2,5179	2,5190

Storm has to load a model from file. In order to parse a PRISM model, one has to give a path to the file with a symbolic description of the model to the stormpy.parse_prism_program(). But it is not possible to parse a PRISM model directly from the string. Though it has to be said that PRISM requires additional parsing of the results.

In addition to Storm vs PRISM comparison, the comparison between different types of PRISM modes was also carried out: explicit, MTBDD, sparse, hybrid and exact. Unfortunately, exact mode didn't work and every time a following type of error showed: "Error: Probabilities sum to 49999995999999/5000000000000 instead of 1". For the testing 100 models have been generated. For every model two properties have been checked. Every model has been checked four times. Therefore, for every model we got 8 checked properties. Mean time of property analysis per every model has been calculated. The snippet of results and a total result of the comparison is presented in Table 2. The comparison has shown that explicit mode of PRISM is the fastest one. The hybrid engine is enabled by default in PRISM [16]. It uses a combination of symbolic and explicit-state data structures (as used in the MTBDD and sparse engines, respectively). The explicit engine is similar to the sparse engine, in that it can be a good option for relatively small models, but will not scale up to some of the models that can be handled by the hybrid or MTBDD engines. Therefore, it is recommended to continue using the hybrid mode of PRISM.

4. Conclusion

An importance of the risk assessment in the systems has been discussed in this article. General description of the DEPM workflow with the focus on the PMC interface has been presented and various reliability metrics along with the example of the advanced reliability properties had been shown. Usage of the PRISM model checker in the context of reliability characteristics evaluation has been illustrated. Storm model checker has been reviewed. In order to evaluate its efficiency and assess the necessity of using Storm instead of PRISM, a set of experiments has been conducted. The article has presented the results of the efficiency comparison of these two model checkers based on the automatically generated set of the DTMC models. Several computation engines that are supported by both model checkers have been taken into account. The review of the Storm model checker has shown that Storm does not support some of the PRISM features, such as: computing of the steady-state and transient probabilities, probabilistic timed automata, and multi-objective model checking. The effectiveness comparison has shown that Storm model checker is almost three times faster than PRISM. But due to the abovementioned restrictions of its functionality, it is not worth to be implemented to OpenErrorPro yet. At least, until these functions will be developed by the authors.

Aknowledgment
This work has been supported by Erasmus+ EU funds.

References

1. Maria Svorenova and Marta Kwiatkowska. Quantitative Verification and Strategy Synthesis for Stochastic Games. European Journal of Control, Elsevier, 2016.
2. A. Morozov and K. Janschek. Dual graph error propagation model for mechatronic system analysis. In 18th IFAC World Congress, Milano, Italy, pages 9893–9898, 2011.
3. A. Morozov, R. Tuk, and K. Janschek. ErrorPro: Software Tool for Stochastic Error Propagation Analysis. In 1st International Workshop on Resiliency in Embedded Electronic Systems, Amsterdam, The Netherlands, pages 59–60, 2015.
4. Fabarisov, T., Yussupova, N., Ding, K., Morozov, A., Janschek, K.: Analytical Toolset for Model-based Stochastic Error Propagation Analysis: Extension and Optimization Towards Industrial Requirements. Proceedings of the 19th international workshop on computer science and information technologies, Germany, Baden-Baden, 2017.
5. A. Morozov and K. Janschek. Probabilistic error propagation model for mechatronic systems. Mechatronics, 24(8):1189 – 1202, 2014.
6. IMC Networks: MTBF, MTTR, MTTF & FIT Explanation of Terms, Aarschot, Belgium, 2011.
7. Marta Kwiatkowska, Gethin Norman, Dave Parker Dave Parker. Probabilistic Model Checking Probabilistic Model Checking, University of Oxford.
8. Vojtěch Forejt, Marta Kwiatkowska, Gethin Norman and David Parker. Automated Verification Techniques for Probabilistic Systems. In M. Bernardo and V. Issarny (editors), Formal Methods for Eternal Networked Software Systems (SFM'11), volume 6659 of LNCS, pages 53-113, Springer. June 2011.
9. Marta Kwiatkowska, Gethin Norman and David Parker. PRISM 4.0: Verification of Probabilistic Real-time Systems. In Proc. 23rd International Conference on Computer Aided Verification (CAV'11), volume 6806 of LNCS, pages 585-591, Springer, 2011.
10. Marta Kwiatkowska, Gethin Norman and David Parker. Probabilistic Verification of Herman's Self-Stabilisation Algorithm. Formal Aspects of Computing, 24(4), pages 661-670, Springer. July 2012.
11. Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, Matthias Volk. A Storm is Coming: A Modern Probabilistic Model Checker. Proc. of CAV, Volume 10427 of LNCS, pages 592–600, Springer, 2017.
12. Yuxin Deng, Matthew Hennessy, On the semantics of Markov automata, Information and Computation, Volume 222, 2013, Pages 139-168, ISSN 0890-5401.
13. Tim Quatmann, Christian Dehnert, Nils Jansen, Sebastian Junges, Joost-Pieter Katoen. Parameter Synthesis for Markov Models: Faster Than Ever. Proc. of ATVA, Volume 9938 of LNCS, pages 50–67, Springer, 2016.
14. Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, Matthias Volk. The Probabilistic Model Checker Storm (Extended Abstract). CoRR abs/1610.08713, 2016.
15. Gethin Norman and David Parker. Quantitative Verification: Formal Guarantees for Timeliness, Reliability and Performance. A report by the London Mathematical Society and the Smith Institute. Edited by Robert Leese and Tom Melham. 2014.
16. Marta Kwiatkowska, Gethin Norman and David Parker. Advances and Challenges of Probabilistic Model Checking. In Proc. 48th Annual Allerton Conference on Communication, Control and Computing, pages 1691-1698, IEEE Press. Invited paper. October 2010.