DiPro - A Tool for Probabilistic Counterexample Generation

Husain Aljazzar, Florian Leitner-Fischer, Stefan Leue, and Dimitar Simeonov

University of Konstanz, Germany

Abstract. The computation of counterexamples for probabilistic model checking has been an area of active research over the past years. In spite of the achieved theoretical results in this field, there is no freely available tool that allows for the computation and representation of probabilistic counterexamples. We present an open source tool called DiPro that can be used with the PRISM and MRMC probabilistic model checkers. It allows for the computation of probabilistic counterexamples for discrete time Markov chains (DTMCs), continuous time Markov chains (CTMCs) and Markov decision processes (MDPs). The computed counterexamples can be rendered graphically.

1 Introduction

Due to the numerical nature of the used model checking algorithm stochastic model checkers are unable to derive a counterexample witnessing a property violation directly from the model checking process. The unavailability of informative counterexamples makes debugging very difficult, which constrains the practical usefulness of current stochastic model checking tools. The provisioning of such counterexamples requires addressing the following crucial issues:

- The definition of the *notion of a counterexample*: Since probability bounds are in most cases not exceeded by single execution sequences, stochastic counterexamples are formed by sets of execution sequences that jointly exceed probability limits [5].
- The *representation* of counterexamples: It has been proposed to either enumerate these execution sequences, or to represent them as graphs or algebraic expressions.
- The *selection of the traces* belonging to the counterexample: Heuristics guided search has successfully been used to select the counterexamples carrying most probability mass first.
- The *presentation* of counterexamples: Due to the large number of traces in the counterexample, a visual representation is required [3]. It has also been proposed to abstract probabilistic counterexamples into Fault Trees [12].

We present a tool, called DiPro, which we have designed and implemented in order to experimentally evaluate our algorithmic approaches towards counterexample generation in stochastic model checking. The tool implements the XBF

algorithms to compute counterexamples as diagnostic subgraphs, using heuristics guided search [5]. Alternatively, it enumerates counterexamples using the K* algorithm, a heuristics guided on-the-fly variant of Eppstein's k-shortest-paths (KSP) algorithm that we developed [6]. When computing counterexamples for Markov Decision Processes (MDPs), DiPro uses an AND/OR tree datastructure based approach [4]. Counterexamples are represented graphically, as proposed in [3]. While there exists a significant body of alternative work on the computation of counterexamples, e.g., [9, 7], we are not aware of any other publicly available counterexample generation tool for CTMCs, DTMCs and MDPs.

2 Probabilistic Counterexamples

We briefly introduce the concept of probabilistic counterexamples, for a more detailed account see [5].

We call a path starting at the initial state of the system and leading to a state satisfying some state formula φ in some stochastic temporal logic, such as CSL [8], a diagnostic path. While in functional model checking a single path often provides valuable information for the debugging of the system, a single path is not sufficient in the probabilistic setting since the violation of a CSL property can hardly ever be traced back to a single diagnostic path. In almost all cases a set of diagnostic paths is needed to provide an accumulated probability mass that violates the specified probabilistic property. In CSL, an upper bounded property, for instance, can be expressed by a formula of the form $P_{\leq p}(\varphi)$. A counterexample for an upper bounded property is now a set X of diagnostic paths such that the accumulated probability of X violates the probability constraint $\leq p$. For a given model and an upper-bounded CSL formula, the set of diagnostic paths is usually not unique, and often very large. We propose the following criteria to be considered in the selection of the diagnostic paths that are to be included in the counterexamples:

- We prefer shorter diagnostic paths, that is paths with fewer transitions, over longer paths. This is justified by the observation that the shorter the path, the easier it is to interpret and retrace it in the model.
- We prefer diagnostic paths with higher probability mass to be included in the counterexample over paths with lower probability. This means that on the one hand we can keep the size of the counterexample small by considering few diagnostic paths that carry high probability each. On the other hand, executions with a high probability are also often more relevant when debugging the model.

These criteria have led us to use heuristics guided explicit state space search on the DTMC, CTMC or MDP models in order to include few, but high probability diagnostic paths into the counterexamples. XBF uses heuritic guidance to incrementally compute a counterexample in the form of a diagnostic subgraph. K* uses the KSP search idea in combination with heuristic search guidance to successively enumerate the most probable, which is an interpretation of shortness notion in KSP, diagnostic paths until the probability bound has been reached.

3 The DiPro Tool

The DiPro tool can be downloaded from the DiPro website[1]. The DiPro tool is open-source and released under the GNU general public license[2].

The DiPro tool can be used in a command line version, with a graphical user interface or as a library that can be invoked via an application program interface. All three version can export the counterexample in a text-file or in an XML-file for later usage. DiPro can be used to compute counterexamples jointly with the PRISM [10] and the MRMC [11] model checking tools. The graphical user interface of DiPro provides a wizard that guides the user from opening a PRISM or MRMC model file via the selection of a stochastic temporal logic formula to the settings of the counterexample computation. Once the wizard is completed, a visualization and counterexample computation window is loaded.

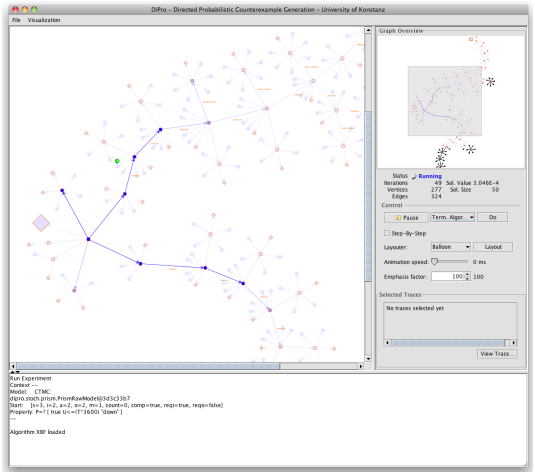

Fig. 1. DiPro graphical user interface.

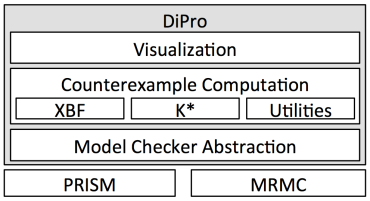

Fig. 2. DiPro architecture.

Figure 1 shows the visualization and counterexample computation window of DiPro. In the upper right part of the window the counterexample is visualized as a graph. The visualization represents the counterexamples as a graph resulting from an unfolding of the CTMC, DTMC or MDP being analyzed. All paths belonging to the counterexample are colored in red, whereas paths not belonging to the counterexample are colored in blue. The states of the paths are represented as circles connected by arrows, which represent the transitions. All end-states of diagnostic paths are depicted as diamonds, the sizes of which correlate with the probability masses of the diagnostic paths ending in these states.

[1] http://www.se.inf.uni-konstanz.de/DiPro
[2] http://www.gnu.org/licenses/

We envision DiPro not only as a stand-alone tool, but as being integrated in a tool chain, which is facilitated by its application program interface. In [12, 13] we show how the counterexamples generated by DiPro can be used by other tools. This way we achieve an integration into the UML modelling world. We also enable counterexamples to be abstracted and represented as Fault Trees.

A pictorial overview of the essentially layered architecture of DiPro is given in Fig. 2. The *Model Checker Abstraction* provides all necessary interfaces to control access to PRISM and MRMC. The algorithms for the counterexample computation, together with data structures needed to store the counterexample and some additional utilities are included in the *Counterexample Computation* layer. The *Visualization* layer comprises everything necessary for the visualization of the counterexamples. DiPro is implemented in JAVA.

4 Case Study: Airbag System

This case study is taken from [1] and models an industrial automotive airbag system. The architecture of this system was provided by our industrial partner TRW Automotive GmbH. The airbag system architecture consists of two acceleration sensors, whose task it is to detect front or rear crashes, one microcontroller to perform the crash evaluation, and an actuator, called FASIC, that controls the deployment of the airbag. Although airbags save lives in crash situations, they may cause fatalities if they are inadvertently deployed. It is therefore a pivotal safety requirement that an airbag is never deployed if there is no crash situation. Suppose that the upper bound for an inadvertent deployment of the airbag within time T is p. In CSL, this property can be expressed using the formula $\mathcal{P}_{<p}(noCrash\ U^{\leq T}\ AirbagIgnited)$. Figure 3 shows one path in the counterexample visualization computed by DiPro. It starts at the initial state (1) and leads to an error state (4) after 3 transitions. We also see that a transition labeled with FASICShortage is the transition leading to the error state, and that the error state has a very high probability. To prevent the inadvertent deployment of the airbag we have to prevent the FASICShortage transition. This can be achieved by replacing the FASIC by an actuator that is more reliable. Extensive experimental evaluations of the algorithms implemented in DiPro can be found in [4, 2, 5, 6].

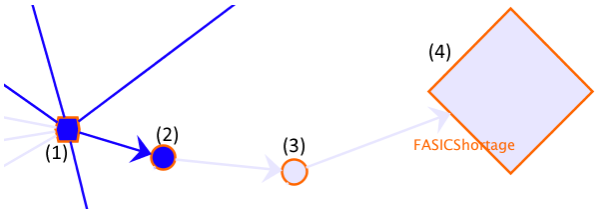

Fig. 3. Excerpt from the airbag system counterexample.

5 Conclusion

We have presented the publicly available tool DiPro for the computation of probabilistic counterexamples for DTMCs, CTMCs and MDPs. In future work we plan to work on automatic generation of heuristics for the counterexample generation algorithms included in DiPro.

References

1. H. Aljazzar, M. Fischer, L. Grunske, M. Kuntz, F. Leitner-Fischer, and S. Leue. Safety analysis of an airbag system using probabilistic FMEA and probabilistic counterexamples. In *Proc. QEST '09*, pages 299–308, Los Alamitos, CA, USA, 2009. IEEE Computer Society.
2. H. Aljazzar, M. Kuntz, F. Leitner-Fischer, and S. Leue. Directed and heuristic counterexample generation for probabilistic model checking: a comparative evaluation. In *Proc. QUOVADIS '10*, pages 25–32, New York, NY, USA, 2010. ACM.
3. H. Aljazzar and S. Leue. Debugging of dependability models using interactive visualization of counterexamples. In *Proc. QEST '08*. IEEE Computer Society Press, 2008.
4. H. Aljazzar and S. Leue. Generation of counterexamples for model checking of markov decision processes. In *Proc. QEST '09*, pages 197–206, Los Alamitos, CA, USA, 2009. IEEE Computer Society.
5. H. Aljazzar and S. Leue. Directed explicit state-space search in the generation of counterexamples for stochastic model checking. *IEEE Transactions on Software Engineering*, 36(1):37–60, 2010.
6. H. Aljazzar and S. Leue. K*: A heuristic search algorithm for finding the k shortest paths. *Artificial Intelligence*, 2011. Accepted for publication.
7. M. Andrés, P. D'Argenio, and P. van Rossum. Significant diagnostic counterexamples in probabilistic model checking. *Hardware and Software: Verification and Testing*, pages 129–148, 2009.
8. C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(7), 2003.
9. T. Han, J. Katoen, and D. Berteun. Counterexample Generation in Probabilistic Model Checking. *IEEE Transactions on Software Engineering*, pages 241–257, 2009.
10. A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. TACAS '06*, Lecture Notes in Computer Science, pages 441–444. Springer, 2006.
11. J.-P. Katoen, M. Khattri, and I. S. Zapreev. A Markov Reward Model Checker. In *QEST '05: Proceedings of the Second International Conference on Quantitative Evaluation of Systems*, pages 243–244. IEEE Computer Society, 2005.
12. M. Kuntz, F. Leitner-Fischer, and S. Leue. From probabilistic counterexamples via causality to fault trees. Technical Report soft-11-02, Chair for Software Engineering, University of Konstanz, 2011. Available from http://www.inf.uni-konstanz.de/soft/research/publications/pdf/soft-11-02.pdf.
13. F. Leitner-Fischer and S. Leue. QuantUM: Quantitative safety analysis of UML models. In *Proc. QAPL 2011*, 2011.

Presentation Plan

We plan to structure the presentation into three parts, first a brief introduction to probabilistic counterexamples. Second an overview of the features of DiPro as well as its architecture, a brief introduction to the used algorithms, and application scenarios of Dipro. And third a demo of the DiPro tool and the counterexample visualization.

1. Part 0: Motivation
2. Part 1 (10%): Introduction to probabilistic Counterexamples
   What distinguishes a counterexample of a functional model checker from a probabilistic counterexample.
3. Part 2 (30%): The DiPro Tool
   - Key Features: Counterexample computation for DTMC, CTMC, MDPs, Visualization
   - Architecture
   - Brief introduction to the algorithms XBF and K*
   - Application scenarios: Debugging, integration of DiPro into a tool chain
4. Part 3 (60%): Tool Demonstration
   In this part, we will demonstrate DiPro based on the Airbag case study discussed in the paper. First, we will show how the graphical user interface wizard guides the user through the different settings needed for the computation. Second, we will show how the computation process of the counterexample can be monitored and controlled. Finally, we will show how the counterexample visualization can be used to glean valuable information for the debugging of the model.