BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2021
http://excel.fit.vutbr.cz

PAYNT: Towards Powerful Automated Synthesis of Probabilistic Programs

Šimon Stupinský*

Abstract
Probabilistic programs play a key role in various engineering domains, including computer networks, embedded systems, power management policies, or software product lines. This paper presents PAYNT, a tool for the automatic synthesis of probabilistic programs satisfying the given specification. It supports the synthesis of finite-state probabilistic programs representing a finite family of candidate programs. PAYNT provides a novel integrated approach for probabilistic synthesis developed on the principles of abstraction refinement (AR) and counterexample-guided inductive synthesis (CEGIS) methods. PAYNT is able to efficiently synthesise the topology of the program as well as continuous parameters affecting the transition probabilities – this is a unique feature. Existing tools for topology synthesis implement only naive approaches and thus typically do not scale to practically relevant synthesis problems. Tools leveraging search-based techniques can handle both synthesis problems, but they do not ensure the complete exploration of the design space. For challenging synthesis problems, PAYNT is able to significantly decreases the run-time from days to minutes while ensuring the completeness of the synthesis process. We demonstrate the usefulness and performance of PAYNT on a wide range of benchmarks from different application domains. Our tool paper presenting PAYNT has been recently accepted at CAV'21, an A* conference.

Keywords: automated synthesis — probabilistic programs — Markov models — model checking

Supplementary Material: Project Repository

*xstupi00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Randomisation is essential to research areas such as *probabilistic programming*, planning (noisy and unknown environments), dependability (system components with uncertainty), and distributed computing (symmetry breaking). Probabilistic programs are a powerful tool for modelling systems that contain probabilistic uncertainty. Their application covers a broad range of research areas, including, e.g. analysis of (quantitative) software product lines [1], strategy synthesis in planning under partial observability [2], or design of communication protocols [3]. During the design, such systems with uncertainty require verification of their correctness and efficiency.

Correctness and efficiency of the probabilistic program is often expressed as a set of declarative temporal constraints. The model checkers for probabilistic systems, such as STORM [4] or PRISM [5], provide automated verification of such constraints under a fixed program structure. However, system designs are prevailingly incomplete at the initial development phases because, in most cases, there are no known all system details or intentionally left out. These undefined system details are referred to as *holes*, and they can represent, e.g., a partially implemented controller for a complex system. Program with holes can be thought of as a *sketch*, naturally fitting the required use cases, which conveniently describes a family of designs. A key aspect of the design cycle is to explore these designs, i.e., to do design space exploration. The synthesis challenge is to identify a concrete system with a fully-defined behaviour that is correct and, potentially, even exhibits optimal behaviour. Additionally, developers do not need to redesign the whole

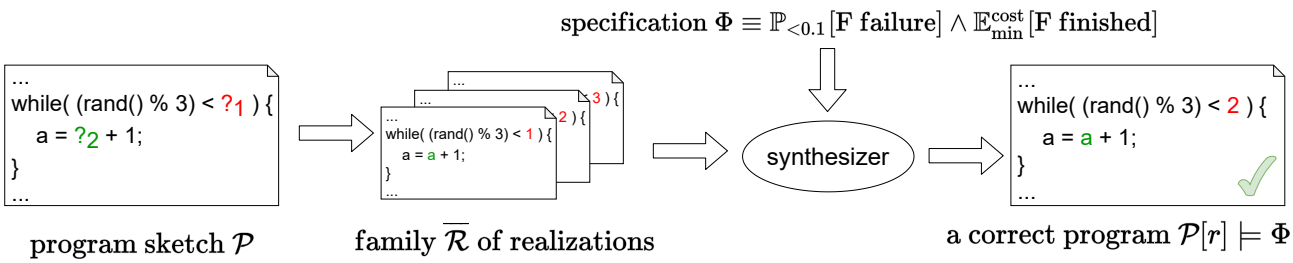

Figure 1. The workflow of the synthesis process within PAYNT.

system when sub-components change because each subsystem is verified against specifications.

Definitely, enumerating all family members (realisations) is unfeasible due to the combinatorial design space explosion. Over the last years, there has been significant algorithmic progress in the analysis of probabilistic programs and temporal logic constraints. Baier *et al.* explored symbolic model-checking methods considering the realisations sets at once [6]. An *inductive* synthesis is a widespread technique based on the idea of counterexamples when we analyse a single realisation and then generalise the results to the realisations set. However, using counterexamples for probabilistic systems is challenging because they usually have a intricate structure. Češka *et al.* [7] used *abstraction refinement* (AR) on realisation set and complemented this with a counterexample-guided inductive synthesis approach (CEGIS) [8]. These two techniques have been recently integrated into a hybrid approach [9] yielding an acceleration of multiple orders of magnitude comparing to the existing approaches.

The main contribution of this work is a new tool called PAYNT (Probabilistic progrAm sYNTthetiser). PAYNT supports the synthesis of finite-state programs and the specifications given as a conjunction of temporal logic constraints, possibly including an optimal objective. The input is a program sketch that concisely describes a finite family of (finite) Markov chains, and a specification. The tool can then identify a family member that (potentially optimally) satisfies given specifications. Figure 1 depicts the workflow of PAYNT. The design of PAYNT is based on an oracle-guided synthesis [9] enabling a flexible combination and integration of a variety of state-of-the-art synthesis and verification algorithms. In particular, it implements a hybrid synthesis approach leveraging both the *CEGIS* and the *AR* oracle. PAYNT is able to efficiently synthesise the program topology (*topology* synthesis) as well as continuous parameters affecting the transition probabilities (*parameter* synthesis). Moreover, it can handle sketches including both types of synthesis problems, so-called a *combined* synthesis, which is a unique feature compering to the existing tools.

To achieve a high-performance synthesis, PAYNT is implemented on top of the probabilistic model checker STORM [4], providing optimised verification procedures. It is implemented in a modular fashion on top of a python API to provide flexibility. PAYNT allows to define of the program sketches and provides all baseline synthesis algorithms under one roof. The tool finds application primarily for two groups of users. First, the analysis of realisations sets is a useful backend for automated system design. For instance, this can be used when synthesising finite-state controllers for partially observable Markov decision processes (POMDPs), synthesising a network protocol to increase the packet throughput, or selecting the optimal power management strategy. Second, PAYNT provides a modular development platform for automated design of probabilistic programs.

2. Related Work

The synthesis tasks for parametric probabilistic programs can be divided into the following two categories.

Topology Synthesis. It considers a finite set of parameters that affect the topology of MCs. Finding a family member that satisfies given specifications, is NP-complete in the parameters' number and can be naively solved by analysing all individual family members [10]. An alternative approach models the family of MCs by an MDP and use off-the-shelf algorithms for MDP model-checking [11]. Tools such as QFLan [12] and ProFeat [13] implements this approach to quantitatively analyse alternative designs of software product lines [1]. These tools provide the complete methods to solve synthesis tasks, but they are limited strictly to small families, so they do not scale. Experiments in previous papers [7, 8] have shown that the synthesis methods implemented in ProFeat or QFLan have clear deficits on the benchmarks investigated in this paper. This fact is supported by the comparison of the hybrid approach with the one-by-one enumeration, we present in Section 5.

Parameter Synthesis. It considers models with a fixed topology but with uncertain parameters associated

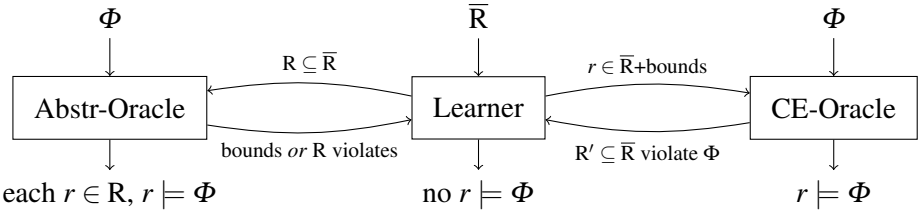

Figure 2. Oracle-guided synthesis approach (adapted from [9])

with transition probabilities (or rates). It analyses how the parameter values affect the behaviour of MC (or MDP). The state-of-the-art probability model checkers STORM [4] and PRISM [5] provide approximate techniques for parameter synthesis that solve the same parameters in various transitions independently. We note that these model checkers and others with the same focus are primarily determined to verify concrete MC and not to the whole MCs family. On the contrary, exact techniques construct rational functions for symbolic reachability probabilities proposed in [14] and further improved in [15]. The application of this synthesis task can be found, for instance, in the field of model repair problems [16].

Search-based approaches can handle both synthesis tasks, but they do not guarantee an exhaustive exploration of the parameter space. Genetic algorithms and evolutionary techniques belong to these approaches group, and the tool RODES [17] implements their combination with parameter synthesis. However, these approaches are not complete and can only effectively solve a feasible instance within the feasibility synthesis task. In the case of an unfeasible instance or optimally synthesis task, they cannot be applied to them. Thus, we omit comparison with the incomplete methods that cannot treat these types of synthesis tasks.

To conclude this section, PAYNT is a unique tool providing complete and scalable methods for the topology, parameter and combine synthesis of probabilistic programs.

3. Implemented Synthesis Methods

Apart from state-of-the-art inductive methods described below, PAYNT also implements the one-by-one approach [10] that enumerates all family members and provides a reference algorithm.

Oracle-Guided Methods. At the heart of PAYNT is an oracle-guided inductive synthesis approach. A *learner* selects a realisation r from the family and passes it to an *oracle*. This unit answers whether realisation r satisfies a given specification Φ. Whenever it is not this case, it provides supplementary information representing typically counterexample (CEGIS) or bounds from MDP model checking (AR). The oracles implemented by PAYNT can be separated into two orthogonal class according to their features:

(i) **Inductive Oracle** (CE): It tries to infer the declarations (counterexamples) about other family members by analysing individual realisation [8].
(ii) **Deductive Oracle** (AR): Abstraction refinement oracle considers more family members at once and then infers the consequences of these members' constructed aggregation [7].

Hybrid Oracle. *Hybrid* approach combines both described AR and CEGIS approaches and switches between them during the synthesis execution. The learner maintains the subfamilies queue $\overline{\mathrm{R}}$ for subsequent processing and decides which oracle is selected based on their previous efficiency. The *CE-Oracle* analyses the family member r and, when it satisfies the given specification Φ, then returns it as a solution. On the other hand, it can generalise the analysed realisation r into a subfamily $\mathrm{R}' \subseteq \overline{\mathrm{R}}$, and the learner can discard it from the whole design space $\overline{\mathrm{R}}$. Moreover, this oracle exploits the MDP bounds when constructing counterexamples, thanks to which it can generate more petite generalisations.

The *Abstr-Oracle* analyses a given sub-family $\mathrm{R} \subseteq \overline{\mathrm{R}}$, and according to the result, it performs the subsequent action. When all realisations $r \in \mathrm{R}$ satisfy Φ, then it returns the overall synthesis result as feasible. In another case, when all realisations violate Φ, the whole analysed sub-family R will be discarded from the families queue $\overline{\mathrm{R}}$ by the learner. The last option returns safe bounds on the best- and worst-case behaviour of all realisations in R considered Φ when the analysis result is inconclusive. Figure 2 depicts described communication between the learner and two oracles.

Hybrid approach primarily performs the loop of abstraction refinement and, whenever it confronts an undecidable family, it switches to the CEGIS approach. It has a chance to analyse the family with its technique for a limited period, and either it finds a solution, it prunes the whole family, or it will be interrupted, and the AR will run again. The constructed counterexamples exclude several family members, which manifests

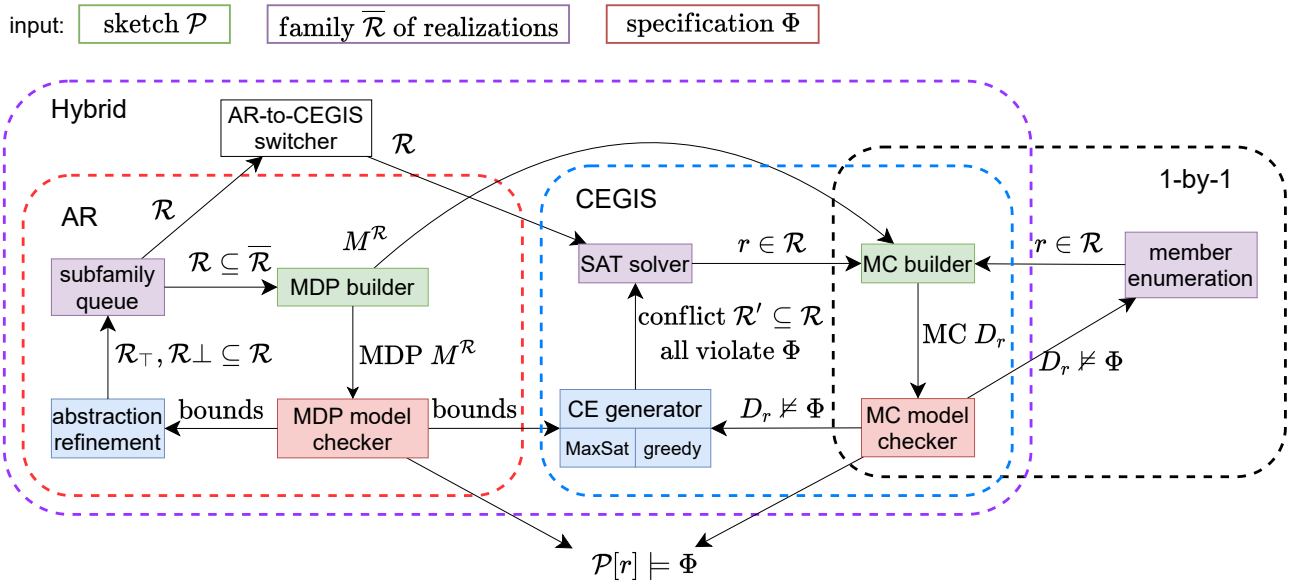

Figure 3. The architecture of tool PAYNT.

itself so that the CEGIS engine updates SAT-formula representing the analysed family to avoid repeated analysis.

3.1 Combined Synthesis

We present the main ideas behind a novel adaptation of the hybrid method towards the *parameter* and *combined* synthesis. Recall that we need to work with parameters having continuous domains and affecting the transition probabilities. We build on a *parameter substitution* [18] that replaces each parametric transition by two non-deterministic choices corresponding to the two extrema of its domain. Applying this operation to a parametric MC yields a parameter-free MDP, which can be verified using the off-the-shelf, efficient algorithms for model checking. We also design a greedy splitting strategy based on the domain size.

Constructing counter-examples for families including continuous parameters requires an additional reasoning as excluding the concrete value from the continuous interval is not feasible. Therefore, we design a strategy trying to exclude a certain part of the interval, which preserves the CE generalisation. For each such parameter, we construct an appropriate neighbourhood of the value in the current assignment – this leads to a MDP model. When the analysis of this MDP yields unsatisfiable results, these neighbourhoods can be safely excluded from interval and thus from the family. Otherwise, the neighbourhoods are gradually refined until the unsatisfiable result is obtained.

3.2 PAYNT Architecture.

PAYNT's architecture (see Figure 3) consists of model checkers, modules to build models and components for family handling. The family handlers store the information about the already covered design space of the analysed family. As the name suggests, the member enumeration unit iterates over all family members. SAT solver maintains an SAT-formula describing undiscovered realisations, and subsequent, it uses Z3 SMT-solver to obtain the next candidate realisation. The queue with sub-families contains a collection of unexplored sub-families refined as hyper-rectangles when the analysis is inconclusive. The model builders take a specific input according to the active oracle and produce the relevant model representation: the CE-oracle sends to the builder a single realisation r while the AR-oracle sends a realisations set $\mathrm{R}' \subseteq \mathrm{R}$. The model checkers verify whether the constructed model (MCs or MDPs) satisfies the given specification. When analysing MDP, lower and upper bounds on satisfiability probabilities are provided. Last but not least, PAYNT provides a module for generating counterexamples. In particular, it implements two approaches: a greedy state-expansion and a MaxSat approach.

Implementation Frame. PAYNT takes as the input a sketch written in the JANI or PRISM language and a set of temporal properties expressed using the PRISM syntax. PAYNT is implemented on top of a modern probabilistic model checker STORM [4] providing high-performance verification procedures implemented in C++. Further, it uses Z3 theorem prover for SMT-solving and a Python API for flexible implementation of the synthesis loop itself.

4. Usage of PAYNT

PAYNT accepts input as a *sketch* in PRISM (or JANI) language containing undefined parameters with associated domains, and specification given as a list of

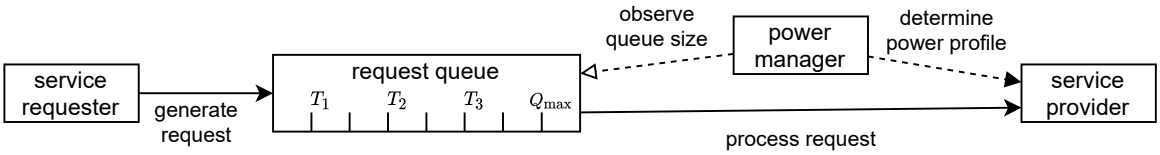

Figure 4. The server for request processing within *DPM* case study.

temporal logic constraints possibly including an optimal objective. A PRISM sketch consists of one or more reactive modules that may interact with each other using synchronisation. A module state space is given by the set of (bounded) variables, and the set of guarded commands, in the following form, describe the possible transitions between states in one module:

$$\texttt{[act]}\ g\ \rightarrow\ p_1 : \texttt{update}_1 + \cdots + p_n : \texttt{update}_n$$

The actions *act* ensure the synchronisation between two or more modules when they perform the command. An update of the variables is selected concerning the probability distribution defined by expressions p_1 through p_n, when guard g evaluates to true. The parameters (holes) can appear in all parts of the command. Replacing each hole with one of its options yields a complete program with the semantics given by a finite-state Markov chain.

We demonstrate the usage of PAYNT on a synthesis problem considering a simple server for request processing. depicted in Figure 4. Requests are produced by an external unit within random intervals and maintained in a request queue with capacity Q_{max}. If the queue is full arriving requests are lost. The server can operate in three modes having a different power consumption – *active*, *idle* and *sleeping*. The server process the requests only in the *active* state. When the server switches from a state with low-energy into a state with higher, then extra energy is consumed, and random latency is required before continuing. We note that the power consumption of request processing depends on the current size of the queue. The server operation time is finite but given as a random process.

The goal of the synthesis is to construct a unit called *power manager* (PM), which controls the server. PM observes the current queue size and, according to it, then sets the relevant power profile. Precisely, it differentiates between four queue occupancy levels determined by the threshold levels T_1, T_2, and T_3. They indicate which fraction of the queue capacity is currently occupied, and they are entered into the model as unknown parameters. Since the model considers three levels, then the power manager observers the queue occupancy on the following intervals: $[0, T_1]$, $(T_1, T_2]$, $(T_2, T_3]$, $(T_3, 1)$. Moreover, it considers a single power profile $P_1, \ldots, P_4 \in \{0, 1, 2\}$ for each occupancy level. The power profile's current value represents the server's mode, so the set $\{0, 1, 2\}$ encodes available modes sleeping, idle and active in the given order. The following sketch describes the PM module.

```
module PM
 // 0 - sleep, 1 - idle, 2 - active
 pm  :  [0..2] init 0;
 [s] q<=T1*QMAX -> (pm'=P1);
 [s] T1*QMAX<q<=T2*QMAX -> (pm'=P2);
 [s] T2*QMAX<q<=T3*QMAX -> (pm'=P3);
 [s] q>T3*QMAX -> (pm'=P4);
endmodule
```

The final sketch describing this considered model forms a design space of 16,200 various power managers. The synthesis target is to find the specific power manager, i.e., the holes instantiation, that minimises power consumption while the expected number of lost requests during the server's operation time is below 1. We can formalise these requirements as a pair of temporal logic formulae in the PRISM language:

```
R{"lost"}<= 1 [ F "finished" ]
R{"power"}min=? [ F "finished" ]
```

The synthesised power manager performs the following strategy. When the request queue is empty, then the power manager maintains an idle state. Otherwise, it always maintains an active state, regardless of the exact size of the queue, and is never in the sleeping state. The synthesised solution has a power consumption of 9,100 units and the expected number of lost requests of $\approx 0.68 < 1$.

PAYNT computes an optimal solution in one minute. Although this synthesis problem is quite simple (recall it includes only 16k candidates), PAYNT is already $3\times$ faster than a naive enumeration of all realisations. Further, we explored a more complex variant of this problem inspired by the known model of a dynamical power manager for complex electronic systems. The synthesis problem is described by the sketch, which covers family around 43M realisations. PAYNT solves this synthesis problem within 10 hours, whereas the naive enumeration takes more than 1 month.

model	number of parameters	family size	average MC size	1-by-1 enumeration	tool performance hard	tool performance easy
DPM	16	43M	3.6k	35 days *	9.3 h	1.1 h
Maze	22	9.4M	0.2k	1.8 days *	1 h	54 min
Herman	7	3.1M	1.1k	1.5 days *	17 min	1.1 min
Pole	17	1.3M	5.6k	1 day *	8.5 min	5 s
Grid	8	65k	1.2k	32 min	37 s	21 s

Table 1. The listing of the results of PAYNT on two variants of five different case-studies from various domains.

5. Performance Evaluation

The aim of this section is to demonstrate the performance of our tool on various synthesis problems from different application domains. In particular, we compare its performance with the one-by-one enumeration representing the baseline algorithm implemented in the existing synthesis tools such as QFLan [12] and ProFeat [13].

All experiments are run on an Ubuntu 19.04 machine with Intel i5-8300H (4 cores at 2.3 GHz) and using up to 8 GB RAM, with all the algorithms being executed single-threaded. Table 1 lists the basic information about the selected sketches that adapt various synthesis problems considered in the previous papers [7, 8, 9]. The table further shows the run-times of the one-by-one enumeration and the hybrid approach implemented in PAYNT. For each sketch, we report the results for two synthesis problems: the column *easy* represents unfeasibility problems, while the column *hard* represents optimal synthesis problems. In all cases, the synthesis methods have to explore the whole design space. The metrics marks with * represent the qualified estimates.

We observe that the performance varies significantly based on the sketch complexity affected by the family size and the average state-space of the candidate programs. The performance of the hybrid approach is also affected by the complexity of the specification. The results, however, clearly demonstrate that our tool significantly outperforms the baseline algorithm (for hard synthesis problems, it reduces the runtime from a day to minutes) and thus fundamentally shifts the scalabilty and applicability of automated synthesis of probabilistic programs.

Combined Synthesis. To demonstrate the impact of the combined synthesis method, we consider a variant of Herman's randomised self-stabilisation protocol [3]. In every round of the protocol, each station with a token must choose whether to pass a token or not. In the original version of the protocol, this choice is resolved on a single biased coin flip. However, we are interested in the synthesis of alternatives. We give each station an additional single bit of memory and the choice between multiple different coin biases. The synthesiser aims to identify the protocol version that minimises stabilisation time.

First, we consider the topology synthesis with 25 different coin biases from the interval $[0.00, 0.25]$. A sketch describing a system with 5 stations yields a family of $3.1M$ programs. PAYNT finds the optimal protocol in around 22 minutes. Refining the topology synthesis by considering 250 coin biases from the same interval makes the synthesis problem intractable as the family size increases about the factor 10k. Second, we consider an alternative formulation of the synthesis problem where we replace these discrete options with continuous transition parameters over the same interval. The combined synthesis method requires around 53 minutes and guarantees the optimal solution over the continuous parameters space.

6. Conclusions

We presented PAYNT, a tool for the automated synthesis of probabilistic programs, which implements state-of-the-art synthesis methods. Additionally, PAYNT is the first tool that supports the combined synthesis including unknown topology as well as transition probabilities. The experiments demonstrate that the tool provides an unprecedented scalability and reduces the run-time for challenging problems from days to minutes. In our future work, we plan to improve the abstraction refinement strategy for the combined synthesis and investigate the construction of counterexamples for MDPs.

Acknowledgements

I would like to thank my advisor RNDr. Milan Češka, Ph.D. and my consultant Ing. Roman Andriuschenko for their guidance during my work on PAYNT. This paper is based on the tool paper[1], which has been recently accepted at CAV'21 conference – the author of this paper is also one of the main co-authors of the tool paper.

[1] PAYNT: A Tool for Inductive Synthesis of Probabilistic Programs

References

[1] Genaina N. Rodrigues, Vander Alves, Vinicius Nunes, Andre Lanna, Maxime Cordy, Pierre Yves Schobbens, Amir Molzam Sharifloo, and Axel Legay. Modeling and verification for probabilistic properties in software product lines. In *Proceedings of IEEE International Symposium on High Assurance Systems Engineering*, volume 2015, pages 173–180. IEEE Computer Society Press.

[2] Gethin Norman, David Parker, and Xueyi Zou. Verification and control of partially observable probabilistic real-time systems. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 240–255. Springer, 2015.

[3] T. Herman. Probabilistic self-stabilization. *Inf. Process. Lett.*, 35(2):63–67, 1990.

[4] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A Storm is coming: A modern probabilistic model checker. In *CAV*, volume 10427 of *LNCS*, pages 592–600. Springer, 2017.

[5] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[6] Philipp Chrszon, Clemens Dubslaff, Sascha Klüppelholz, and Christel Baier. ProFeat: feature-oriented engineering for family-based probabilistic model checking. *Formal Asp. Comput.*, 30(1):45–75, 2018.

[7] Milan Češka, Nils Jansen, Sebastian Junges, and Joost-Pieter Katoen. Shepherding hordes of markov chains. In *TACAS (2)*, volume 11428 of *LNCS*, pages 172–190. Springer, 2019.

[8] Milan Češka, Christian Hensel, Sebastian Junges, and Joost-Pieter Katoen. Counterexample-driven synthesis for probabilistic program sketches. In *FM*, volume 11800 of *LNCS*, pages 101–120. Springer, 2019.

[9] Roman Andriushchenko, Milan Češka, Sebastian Junges, and Joost-Pieter Katoen. Inductive synthesis for probabilistic programs reaches new horizons. In *TACAS*, Lecture Notes in Computer Science. Springer, 2021. (to appear, see https://arxiv.org/abs/2101.12683).

[10] Andreas Classen, Maxime Cordy, Patrick Heymans, Axel Legay, and Pierre-Yves Schobbens. Model checking software product lines with SNIP. *Int. J. on Softw. Tools for Technol. Transf.*, 14:589–612, 2012.

[11] Andreas Classen, Maxime Cordy, Patrick Heymans, Axel Legay, and Pierre Yves Schobbens. Formal semantics, modular specification, and symbolic verification of product-line behaviour. *Science of Computer Programming*, 80:416–439, 02 2014.

[12] Andrea Vandin, Maurice H. ter Beek, Axel Legay, and Alberto Lluch Lafuente. Qflan: A tool for the quantitative analysis of highly reconfigurable systems. In Klaus Havelund, Jan Peleska, Bill Roscoe, and Erik de Vink, editors, *Formal Methods*, pages 329–337, Cham, 2018. Springer International Publishing.

[13] Philipp Chrszon, Clemens Dubslaff, Sascha Klüppelholz, and Christel Baier. Profeat: feature-oriented engineering for family-based probabilistic model checking. *Formal Aspects of Computing*, 30, 08 2017.

[14] Conrado Daws. Symbolic and parametric model checking of discrete-time Markov chains. In *ICTAC*, volume 3407 of *LNCS*, pages 280–294. Springer, 2004.

[15] Christian Dehnert, Sebastian Junges, Nils Jansen, Florian Corzilius, Matthias Volk, Harold Bruintjes, Joost-Pieter Katoen, and Erika Ábrahám. PROPhESY: A PRObabilistic ParamEter SYNthesis Tool. In *CAV'15*, volume 9206 of *LNCS*, pages 214–231. Springer, 2015.

[16] Ezio Bartocci, Radu Grosu, Panagiotis Katsaros, C. R. Ramakrishnan, and Scott A. Smolka. Model repair for probabilistic systems. In *TACAS'11*, volume 6605 of *LNCS*, pages 326–340, 2011.

[17] Radu Calinescu, Milan Češka, Simos Gerasimou, Marta Kwiatkowska, and Nicola Paoletti. RODES: A robust-design synthesis tool for probabilistic systems. In *QEST*, pages 304–308. Springer, 2017.

[18] Tim Quatmann, Christian Dehnert, Nils Jansen, Sebastian Junges, and Joost-Pieter Katoen. Parameter synthesis for markov models: Faster than ever. *CoRR*, abs/1602.05113, 2016.