Inductive Synthesis for Probabilistic Programs Reaches New Horizons*

Roman Andriushchenko[1], Milan Češka (✉)[1], Sebastian Junges[2], and Joost-Pieter Katoen[3]

[1] Brno University of Technology, Brno, Czech Republic
ceskam@fit.vutbr.cz
[2] University of California, Berkeley, USA
[3] RWTH Aachen University, Aachen, Germany

Abstract. This paper presents a novel method for the automated synthesis of probabilistic programs. The starting point is a program sketch representing a finite family of finite-state Markov chains with related but distinct topologies, and a reachability specification. The method builds on a novel inductive oracle that greedily generates counter-examples (CEs) for violating programs and uses them to prune the family. These CEs leverage the semantics of the family in the form of bounds on its best- and worst-case behaviour provided by a deductive oracle using an MDP abstraction. The method further monitors the performance of the synthesis and adaptively switches between inductive and deductive reasoning. Our experiments demonstrate that the novel CE construction provides a significantly faster and more effective pruning strategy leading to an accelerated synthesis process on a wide range of benchmarks. For challenging problems, such as the synthesis of decentralized partially-observable controllers, we reduce the run-time from a day to minutes.

1 Introduction

Background and motivation. Controller synthesis for Markov decision processes (MDPs [35]) and temporal logic constraints is a well-understood and tractable problem, with a plethora of mature tools providing efficient solving capabilities. However, the applicability of these controllers to a variety of systems is limited: Systems may be decentralized, controllers may not be able to observe the complete system state, cost constraints may apply, and so forth. Adequate operational models for these systems exist in the form of *decentralized partially-observable MDPs* (DEC-POMDPs [33]). The controller synthesis problem for these models is undecidable [30], and tool support (for verification tasks) is scarce.

This paper takes a different approach: the controller together with the environment can be modelled as probabilistic program sketches where "holes" in the probabilistic program model choices that the controller may make. Conceptually, the controllers of the DEC-POMDP are described by a user-defined finite

* This work has been partially supported by the Czech Science Foundation grant GJ20-02328Y and the ERC AdG Grant 787914 FRAPPANT, the NSF grants 1545126 (VeHICaL) and 1646208, by the DARPA Assured Autonomy program, by Berkeley Deep Drive, and by Toyota under the iCyPhy center.

J. F. Groote and K. G. Larsen (Eds.): TACAS 2021, LNCS 12651, pp. 191–209, 2021.
https://doi.org/10.1007/978-3-030-72016-2_11

family $\mathcal{M}$ of Markov chains. *The synthesis problem that we consider is to find a Markov chain M (i.e., a probabilistic program) in the family $\mathcal{M}$, such that $M \models \varphi$, where φ is the specification.* To allow efficient algorithms, the family must have some structure. In particular, in our setting, the family is parameterized by a set of discrete *parameters* K; an assignment $K \to V$ of these parameters with concrete values V from its associated domain yields a family member, i.e., a Markov chain (MC). Such a parameterization is naturally obtained from the probabilistic program sketch, where some constants (or program parts) can be left open. The search for a family member can thus be considered as the search for a hole-assignment. This approach fits within the realm of syntax-guided synthesis [2].

Motivating example. Herman's protocol [24] is a well-studied randomized distributed algorithm aimed to obtain fast stabilization on average. In [26], a family $\mathcal{M}$ of MCs is used to model different protocol instances. They considered each instance separately, and found which of the controllers for Herman's protocol performs best. Let us consider the protocol in a bit more detail: It considers self-stabilization of a unidirectional ring of network stations where all stations have to behave similarly—an anonymous network. Each station stores a single bit, and can read the internal bit of one (say left) neighbour. To achieve stabilization, a station for which the two legible bits coincide updates its own bit based on the outcome of a coin flip. The challenge is to select a controller that flips this coin with an optimal bias, i.e., minimizing the expected time until stabilization. In a setting where the probabilities range over $0.1, 0.2, \ldots, 0.9$, this results in analyzing nine different MCs. Does the expected time until stabilization reduce if the controllers are additionally allowed to have a single bit of memory? In every step, there are 9·9 combinations for selecting the coin flip and for each memory cell and coin flip outcome, the memory can now be updated, yielding 2·2·2 possibilities. This one-bit extension thus results in a family of 648 models. If, in addition, one allows stations to make decisions depending on the token-bits, both the coin flips and the memory updates are multiplied by a factor 4, yielding $10,368$ models. Eventually, analyzing all individual MCs is infeasible.

Oracle-guided synthesis. To tackle the synthesis problem, we introduce an *oracle-guided inductive synthesis* approach [25,39]. A learner selects a family member and passes it to the oracle. The oracle answers whether the family member satisfies φ, and crucially, gives additional information in case this is not the case. Inspired by [9], if the family member violates the specification φ, our oracle returns a set K' of parameters such that all family members obtained by changing only the values assigned to K' violate φ. We argue that such an oracle must (1) induce little overhead in providing K', (2) be aware of the existence of parameters in the family, and (3) have (resemblance of) awareness about the semantics of the parameters and their values.

Oracles. With these requirements in mind, we construct a counterexample (CE)-based oracle from scratch. We do so by carefully exploiting existing methods. We construct critical subsystems as CEs [1]. Critical subsystems are parts of

the MC that suffice to refute the specification. If a hole is absent in a CE, its value is irrelevant. To avoid the cost of finding optimal CEs—an NP-hard problem [19]—we consider greedy CEs that are similar to [9]. However, our greedy CEs are aware of the parameters, and try to limit the occurrence of parameters in the CE. Finally, to provide awareness of the semantics of parameter values, we provide lower and upper bounds on all states: Their difference indicates how much varying the value at a hole may change the overall reachability probability. These bounds are efficiently computed by another oracle. This oracle analyses a quotient MDP obtained by employing an abstraction method that is part of the abstraction-refinement loop in [10].

A hybrid variant. The two oracles are significantly different. Abstraction refinement is *deductive*: it argues about single family members by considering (an aggregation of) all family members. The critical subsystem oracle is *inductive*: by examining a single family member, it infers statements about other family members. This suggests a middle ground: a *hybrid strategy* monitors the performance of the two oracles during the synthesis and suggests their best usage. More precisely, the hybrid strategy integrates the counterexample-based oracle into the abstraction-refinement loop.

Major results. We present a novel and dedicated oracle deployed in an efficacious synthesis loop. We use model-checking results on an abstraction to tailor smaller CEs. Our greedy and family-aware CE construction is substantially faster than the use of optimal CEs. Together, these two improvements yield CEs that are on par with optimal CEs, but are found much faster. The integration of multiple abstraction-refinement steps yields a superior performance:x We compare our performance with the abstraction-refinement loop from [10] using benchmarks from [10]. Benchmarks can be classified along two dimensions: (A) Benchmarks with a structure good for CE-generation. (B) Benchmarks with a structure good for abstraction-refinement. A-benchmarks are a natural strength of our novel oracle. Our simple, efficient hybrid strategy significantly outperforms the state-of-the-art on A-benchmarks, while it only yields limited overhead for B-benchmarks. Most importantly, the novel hybrid strategy can solve benchmarks that are out of reach for pure abstraction-refinement or pure CE-based reasoning. In particular, our hybrid method is able to synthesize the optimal Herman protocol with memory—the synthesis time on a design space with 3.1 millions of candidate programs reduces from a day to minutes.

Related work The synthesis problems for parametric probabilistic systems can be divided into the following two categories.

Topology synthesis, akin to the problem considered in this paper, assumes a finite set of parameters affecting the MC topology. Finding an instantiation satisfying a reachability property is NP-complete in the number of parameters [12], and can naively be solved by analyzing all individual family members. An alternative is to model the MC family by an MDP and resort to standard MDP model-checking algorithms. Tools such as ProFeat [13] or QFLan [40] take this approach

to quantitatively analyze alternative designs of software product lines [21,28]. These methods are limited to small families. This motivated (1) *abstraction-refinement* over the MDP representation [10], and (2) *counterexample-guided inductive synthesis* (CEGIS) for MCs [9], mentioned earlier. The alternative problem of sketching for probabilistic programs that fit given data is studied, e.g., in [32,38].

Parameter synthesis considers models with uncertain parameters associated to transition probabilities, and analyses how the system behaviour depends on the parameter values. The most promising techniques are based on *parameter lifting* that treats identical parameters in different transitions independently [8,36] and has been implemented in the state-of-the-art probabilistic model checkers Storm [18] and PRISM [27]. An alternative approach based on building rational functions for the satisfaction probability has been proposed in [15] and further improved in [22,17,4]. This approach has been also applied to different problems such as model repair [5,34,11].

Both synthesis problems can be also attacked by *search-based techniques* that do not ensure an exhaustive exploration of the parameter space. These include evolutionary techniques [23,31] and genetic algorithms [20]. Combinations with parameter synthesis have been used [7] to synthesize robust systems.

2 Problem Statement

We formalize the essential ingredients and the problem statement. See [3] for more material.

Sets of Markov chains. A *(discrete) distribution* over a finite set X is a function $\mu\colon S \to [0,1]$ s.t. $\sum_x \mu(x) = 1$. The set $Distr(X)$ contains all distributions over X. The *support* of $\mu \in Distr(X)$ is $\mathrm{supp}(\mu) = \{x \in X \mid \mu(x) > 0\}$.

Definition 1 (MC). *A* Markov chain (MC) *is a tuple* $D = (S, s_0, \boldsymbol{P})$*, where* S *is a finite set of* states*,* $s_0 \in S$ *is an* initial state*, and* $\boldsymbol{P}\colon S \to Distr(S)$ *is a* transition probability function*. We write* $\boldsymbol{P}(s,t)$ *to denote* $\boldsymbol{P}(s)(t)$*. The state* s *is* absorbing *if* $\boldsymbol{P}(s,s) = 1$*.*

Let K denote a finite set of discrete parameters with finite domain V_k. For brevity, we often assume that all domains are the same, and omit the subscript k. A *realization* r maps parameters to values in their domain, i.e., $r\colon K \to V$. Let $\mathcal{R}^{\mathcal{D}}$ denote the set of all realizations of a set $\mathcal{D}$ of MCs. A K-parameterized set of MCs $\mathcal{D}(K)$ contains the MCs $\mathcal{D}_r$, for every $r \in \mathcal{R}^{\mathcal{D}}$. In Sect. 3, we give an operational model for such sets. In particular, realizations will fix the targets of transitions. In our experiments, we describe these sets using the PRISM modelling language where parameters are described by undefined integer values.

Properties and specifications. For simplicity, we consider (unbounded) *reachability* properties[1]. For a set $T \subseteq S$ of *target states*, let $\mathbb{P}[D, s \models \Diamond T]$ denote

[1] Our implementation also supports expected reachability rewards.

the probability in MC D to eventually reach some state in T when starting in the state $s \in S$. A property $\varphi \equiv \mathbb{P}_{\bowtie\lambda}[\Diamond T]$ with $\lambda \in [0,1]$ and $\bowtie \in \{\leq, \geq\}$ expresses that the probability to reach T does relate to λ according to $\bowtie$. If $\bowtie = \leq$, then φ is a *safety* property; otherwise, it is a *liveness* property. Formally, state s in MC D satisfies φ if $\mathbb{P}[D, s \models \Diamond T] \geq \lambda$. The MC D satisfies φ if the above holds for its initial state. A *specification* is a set of properties $\Phi = \{\varphi_i\}_{i \in I}$, and $D \models \Phi$ if $\forall i \in I : D \models \varphi_i$.

Problem statement. The key problem statement in this paper is *feasibility*:

> Given a parameterized set of Markov chains $\mathcal{D}(K)$ over parameters K and a specification Φ, find a realization $r \colon K \to V$ such that $\mathcal{D}_r \models \Phi$.

When $\mathcal{D}$ is clear from the context, we often write $r \models \Phi$ to denote $\mathcal{D}_r \models \Phi$.

We additionally consider the optimizing variant of the synthesis problem. The *maximal synthesis* problem asks: given a maximizing property $\varphi_{\max} \equiv \mathbb{P}_{\bowtie\lambda}[\Diamond T]$, identify $r^* \in \arg\max_{r \in \mathcal{R}^{\mathcal{D}}} \{\mathbb{P}[\mathcal{D}_r \models \Diamond T] \mid \mathcal{D}_r \models \Phi\}$ provided it exists. The *minimal synthesis* problem is defined analogously.

As the state space S, the set K of parameters, and their domains are all finite, the above synthesis problems are decidable. One possible solution, called the *one-by-one approach* [14], considers each realization $r \in \mathcal{R}^{\mathcal{D}}$. The state-space and parameter-space explosion renders this approach unusable for large problems, necessitating the usage of advanced techniques that exploit the family structure.

3 Counterexample-Guided Inductive Synthesis

In this section, we recap a baseline for a counterexample-guided inductive synthesis (CEGIS) loop, as put forward in [9]. In particular, we first instantiate an oracle-guided synthesis method, discuss an operational model for families, giving structure to the parameterized set of Markov chains, and finally detail the usage of CEs to create an oracle.

Consider Fig. 1. A learner takes a set $\mathcal{R}$ of realizations, and has to find a realization $\mathcal{D}_r$ satisfying the specification Φ. The learner maintains (a symbolic representation of) a set $Q \subseteq \mathcal{R}$ of realizations that need to be checked. It iteratively asks the oracle whether a particular $r \in Q$ is a solution. If it is a solution, the oracle reports success. Otherwise, the oracle returns a set $\mathcal{R}'$ containing r and potentially more realizations all violating Φ. The learner then prunes $\mathcal{R}'$ from Q. In Section 4, we focus on creating an efficient oracle that computes a set $\mathcal{R}'$ (with $r \in \mathcal{R}'$) of realizations that are all violating Φ. In Section 5, we provide a more advanced framework that extends this method. The remainder of this section lays the groundwork for these sections.

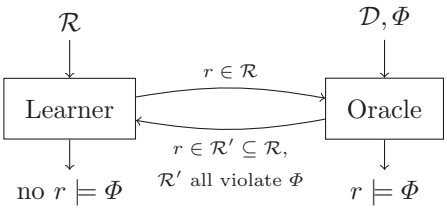

Fig. 1. Oracle-guided synthesis

Families of Markov chains To avoid the need to iterate over all realizations, an efficient oracle exploits some structure of the family. In this paper, we focus on sets of Markov chains having different topologies. We explain our concepts using the operational model of families given in [10]. Our implementation supports (more expressive) PRISM programs with undefined integer constants.

Definition 2 (Family of MCs). *A* family of MCs *is a tuple* $\mathcal{D} = (S, s_0, K, \mathcal{B})$ *with* S *and* s_0 *as before,* K *is a finite set of parameters with domains* $V_k \subseteq S$ *for each* $k \in K$*, and* $\mathcal{B} : S \to Distr(K)$ *is a family of transition probability functions.*

Function $\mathcal{B}$ of a family $\mathcal{D}$ of MCs maps each state to a distribution over parameters K. In the context of the synthesis of probabilistic models, these parameters represent unknown options or features of a system under design. Realizations are now defined as follows.

Definition 3 (Realization). *A* realization *of a family* $\mathcal{D} = (S, s_0, K, \mathcal{B})$ *of MCs is a function* $r : K \to S$ *s.t.* $r(k) \in V_k$*, for all* $k \in K$*. We say that realization* r *induces MC* $\mathcal{D}_r = (S, s_0, \mathcal{B}_r)$ *iff* $\mathcal{B}_r(s, s') = \sum_{k \in K, r(k)=s'} \mathcal{B}(s)(k)$ *for any pair of states* $s, s' \in S$*. The set of all realizations of* $\mathcal{D}$ *is denoted as* $\mathcal{R}^{\mathcal{D}}$*.*

The set $\mathcal{R}^{\mathcal{D}} = \prod_{k \in K} V_k$ of all possible realizations is exponential in $|K|$.

Counterexample-guided oracles We first consider the feasibility synthesis for a single-property specification and later, cf. Remark 1, generalize this to multiple properties and to optimal synthesis. The notion of counterexamples is at the heart of the oracle from [9] and Sect. 4.

If an MC $D \not\models \varphi$, a *counterexample* (CE) based on a critical subsystem can serve as diagnostic information about the source of the failure. We consider the following CE, motivated by the notion of critical subsystem in [37].

Definition 4 (Counterexample). *Let* $D = (S, s_0, \boldsymbol{P})$ *be an MC with* $s_\bot \notin S$*. The* sub-MC *of* D *induced by* $C \subseteq S$ *is the MC* $D{\downarrow}C = (S \cup \{s_\bot\}, s_0, \boldsymbol{P'})$*, where the transition probability function* $\boldsymbol{P'}$ *is defined by:*

$$\boldsymbol{P'}(s) = \begin{cases} \boldsymbol{P}(s) & \text{if } s \in C, \\ [s_\bot \mapsto 1] & \text{otherwise.} \end{cases}$$

The set C *and the sub-MC* $D{\downarrow}C$ *are called a* counterexample *(CE) for the property* $\mathbb{P}_{\leq \lambda}[\Diamond T]$ *on MC* D*, if* $D{\downarrow}C \not\models \mathbb{P}_{\leq \lambda}[\Diamond(T \cap (C \cup \{s_0\}))]$*.*

Let $\mathcal{D}_r$ be an MC violating the specification φ. To compute other realizations violating φ, the oracle computes a critical subsystem $\mathcal{D}_r{\downarrow}C$, which is then used to deduce a so-called *conflict* for $\mathcal{D}_r$ and φ.

Definition 5 (Conflict). *For family of MCs* $\mathcal{D} = (S, s_0, K, \mathcal{B})$ *and* $C \subseteq S$*, the set* K_C *of* relevant parameters *(called* conflict*) is given by* $\bigcup_{s \in C} \mathrm{supp}(\mathcal{B}(s))$*.*

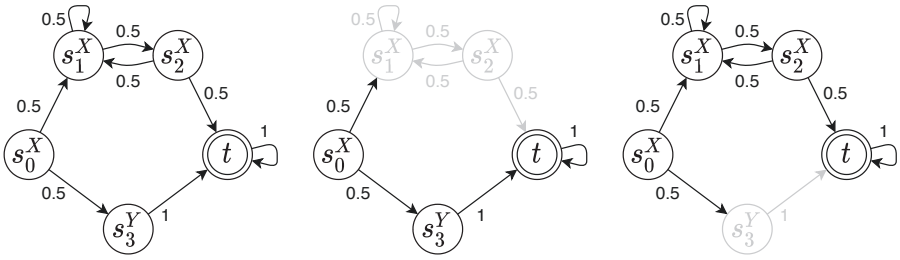

Fig. 2. Counterexamples for smaller conflicts.

It is straightforward to compute a set of violating realizations from a conflict. A *generalization* of realization r induced by the set $K_C \subseteq K$ of relevant parameters is the set $r{\uparrow}K_C = \{r' \in \mathcal{R} \mid \forall k \in K_C : r(k) = r'(k)\}$. We often use the term *conflict* to refer to its generalization. The size of a conflict, i.e., the number $|K_C|$ of relevant parameters K_C is crucial. Small conflicts potentially lead to generalizing r to larger subfamilies $r{\uparrow}K_C$. It is thus important that the CEs contain as few parameterized transitions as possible. The size of a CE in terms of the number of states is not of interest. Furthermore, the overhead of providing CEs should be bounded from below by the payoff: Finding a large generalization may take some time, but small generalizations should be returned quickly. The CE-based oracle in [9] uses an off-the-shelf CE procedure [16,41], and mostly does not provide small CEs.

4 A Smart Oracle with Counterexamples and Abstraction

This section develops an oracle based on CEs, tailored for the use in an oracle-guided inductive synthesis loop described in Sect. 3. Its main features are:

- a fast greedy approach to compute CEs that provide small conflicts: We achieve this by taking into account the position of the parameters.
- awareness about the semantics of parameters by using model-checking results from an abstraction of the family.

Before going into details, we provide some illustrative examples.

A motivating example First, we illustrate what it means to take CEs that lead to small conflicts. Consider Fig. 2, with a family member $\mathcal{D}_r$ (left), where the superscript of a state identifier s_i denotes parameters relevant to s_i. Consider the safety property $\varphi \equiv \mathbb{P}_{\leq 0.4}[\Diamond\{t\}]$. Clearly, $\mathcal{D}_r \not\models \varphi$, and we can construct two CEs: $C_1 = \{s_0, s_3, t\}$ (center) and $C_2 = \{s_0, s_1, s_2, t\}$ (right) with conflicts $K_{C_1} = \{X, Y\}$ and $K_{C_2} = \{X\}$, respectively. It illustrates that a smaller CE does not necessarily induce a smaller conflict.

We now illustrate awareness of the semantics of parameters. Consider the family $\mathcal{D} = (S, s_0, K', \mathcal{B})$, where $S = \{s_0, s_1, s_2, t, f\}$, the parameters are $K' = \{X, Y, T', F'\}$ with domains $V_X = \{s_1, s_2\}$, $V_Y = \{t, f\}$, $V_{T'} = \{t\}$, $V_{F'} = \{f\}$, and a family $\mathcal{B}$ of transition probability functions defined in Fig. 3 (left). As the

$$\mathcal{B}(s_0) = [X \mapsto 1],$$
$$\mathcal{B}(s_1) = [T' \mapsto 0.6, Y \mapsto 0.2, F' \mapsto 0.2],$$
$$\mathcal{B}(s_2) = [T' \mapsto 0.2, Y \mapsto 0.2, F' \mapsto 0.6],$$
$$\mathcal{B}(t) = [T' \mapsto 1],$$
$$\mathcal{B}(f) = [F' \mapsto 1]$$

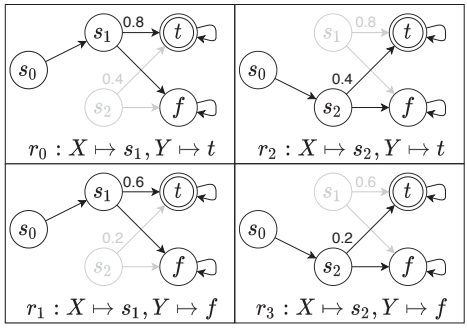

Fig. 3. A family $\mathcal{D}$ of four Markov chains (unreachable states are grayed out).

parameters T' and F' each can take only one value, we consider $K = \{X, Y\}$ as the set of parameters. There are $|V_X| \times |V_Y| = 4$ family members, depicted in Fig. 3(right). For conciseness, we omit some of the transition probabilities (recall that transition probabilities sum to one). Only realization r_3 satisfies the safety property $\varphi \equiv \mathbb{P}_{\leq 0.3}[\Diamond\{t\}]$.

CEGIS [9] illustrated: Consider running CEGIS, and assume the oracle gets realization r_0 first. A model checker reveals $\mathbb{P}[\mathcal{D}_{r_0}, s_0 \models \Diamond T] = 0.8 > 0.3$. The CE for $\mathcal{D}_{r_0}$ and φ contains the (only) path to the target: $s_0 \to s_1 \to t$ having probability $0.8 > 0.3$. The corresponding CE $C = \{s_0, s_1, t\}$ induces the conflict $K_C = \{X, Y\}$. None of the parameters is generalized. The same argument applies to any subsequent realization: the constructed CEs do not allow for generalization, the oracle returns only the passed realization, and the learner keeps iterating until accidentally guessing r_3.

Can we do better? To answer this, consider CE generation as a game: The Pruner creates a critical subsystem C. The Adversary wins if it finds a MC satisfying φ containing C, thus refuting that C is a counterexample. In our setting, we change the game: The Adversary must select a family member rather than an arbitrary MC. Analogously, off-the-shelf CE generators construct a critical subsystem C that for every possible extension of C is a CE. These are *CEs without context*. In our game, the Adversary may not extend the MC arbitrarily, but must choose a family member. These are *CEs modulo a family*.

Back to the example: Observe that for a CE for $\mathcal{D}_{r_0}$, we could omit states t and s_1 from the set C of critical states: we know for sure that, once $\mathcal{D}_{r_0}$ takes transition (s_0, s_1), it will reach target state t with probability at least 0.6. This exceeds the threshold 0.3, regardless of the value of the parameter Y. Hence, for family $\mathcal{D}$, the set $C' = \{s_0\}$ is a critical subsystem. The immediate advantage is that this set induces conflict $K_{C'} = \{X\}$ (parameter Y has been generalized). This enables us to reject all realizations from the set $r_0 \uparrow K_{C'} = \{r_0, r_1\}$. *It is 'easier' to construct a CE for a (sub)family than for arbitrary MCs.* More generally, a successful oracle needs to have access to useful bounds, and effectively integrate them in the CE generation.

Counterexample construction We develop an algorithm using bounds on reachability probabilities, similar to the bounds used above. Let us assume that for some set of realizations $\mathcal{R}$ and for every state s, we have bounds $lb^{\mathcal{R}}(s)$, $ub^{\mathcal{R}}(s)$, such that for every $r \in \mathcal{R}$ we have $lb^{\mathcal{R}}(s) \leq \mathbb{P}[\mathcal{D}_r, s \models \Diamond T] \leq ub^{\mathcal{R}}(s)$. Such bounds always exist (take 0 and 1). We see later how we compute these bounds. In what follows, we fix r and denote $\mathcal{D}_r = (S, s_0, \boldsymbol{P})$. Let us assume $\mathcal{D}_r$ violates a safety property $\varphi \equiv \mathbb{P}_{\leq\lambda}[\Diamond T]$. The following definition is central:

Definition 6 (Rerouting). *Let MC* $D = (S, s_0, \boldsymbol{P})$ *with* $s_\top, s_\bot \notin S$, $C \subseteq S$ *a set of* expanded states *and* $\boldsymbol{\gamma}\colon S \setminus C \to [0,1]$ *a* rerouting vector. *The* rerouting *of MC* D *w.r.t.* C *and* $\boldsymbol{\gamma}$ *is the MC* $D{\downarrow}C[\boldsymbol{\gamma}] = (S \cup \{s_\bot, s_\top\}, s_0, \boldsymbol{P}^C_{\boldsymbol{\gamma}})$ *with:*

$$\boldsymbol{P}^C_{\boldsymbol{\gamma}}(s) = \begin{cases} \boldsymbol{P}(s) & \text{if } s \in C, \\ [s_\top \mapsto \boldsymbol{\gamma}(s), s_\bot \mapsto (1-\boldsymbol{\gamma}(s))] & \text{if } s \in S\setminus C, \\ [s \mapsto 1] & \text{if } s \in \{s_\top, s_\bot\}. \end{cases}$$

Essentially, $D{\downarrow}C[\boldsymbol{\gamma}]$ extends the MC D with additional *sink states* $s_\top$ and $s_\bot$ and replaces all outgoing transitions of any non-expanded state $s \in S\setminus C$ by a transition leading to $s_\top$ (with probability $\boldsymbol{\gamma}(s)$) and a complementary one to $s_\bot$. We consider $s_\top$ to be the new target and let φ' denote the updated property. The transition $s \xrightarrow{\boldsymbol{\gamma}(s)} s_\top$ may be considered a 'shortcut' that by-passes successors of s and leads straight to target $s_\top$ with probability $\boldsymbol{\gamma}(s)$. To ensure that $D{\downarrow}C[\boldsymbol{\gamma}]$ is a CE, the value $\boldsymbol{\gamma}(s)$ must be a lower bound on the reachability probability from s in D. When constructing a CE for a singular MC, we pick $\boldsymbol{\gamma} = \mathbf{0}$, whereas when this MC is induced by a realization $r \in \mathcal{R}$, we can safely pick $\boldsymbol{\gamma} = lb^{\mathcal{R}}$. The CE will be valid for every $r' \in \mathcal{R}$. It is a CE-modulo-$\mathcal{R}$.

Algorithmically, we employ a state-exploration approach and therefore start with $C^{(0)} = \emptyset$, i.e., all states are initially rerouted. If this is a CE, we are done. Otherwise, if the rerouting $D{\downarrow}C^{(0)}[\boldsymbol{\gamma}]$ satisfies φ', then we 'expand' some states to obtain a CE. Naturally, we must expand reachable states to change the satisfaction of φ. By expanding some state $s \in S$, we abandon the abstraction associated with the shortcut $s \xrightarrow{\boldsymbol{\gamma}(s)} s_\top$ and replace it with concrete behavior that was inherent to state s in MC D. Expanding a state cannot decrease the induced reachability probability as $lb^{\mathcal{R}}$ is a valid lower bound. This gradual expansion of the reachable state space continues until for some $C \subseteq S$ the corresponding rerouting $D{\downarrow}C[\boldsymbol{\gamma}]$ violates φ'. This gradual expansion process terminates as $D{\downarrow}S[\boldsymbol{\gamma}] \equiv D$ and our assumption is $D \not\models \varphi$. We show this process on an example.

Example 1. Reconsider $\mathcal{D}$ in Fig. 3 with $\varphi \equiv \mathbb{P}_{\leq 0.3}[\Diamond\{t\}]$. Using the method outlined below we get: $lb^{\mathcal{R}} = [s_0 \mapsto 0.2, s_1 \mapsto 0.6, s_2 \mapsto 0.2, t \mapsto 1, f \mapsto 0]$. In absence of any bounds, the CE is $\{s_0, s_1, t\}$. Consider the gradual rerouting approach: We set $\boldsymbol{\gamma} = lb^{\mathcal{R}}$, $C^{(0)} = \emptyset$ and have $D^{(0)} := \mathcal{D}_{r_0}{\downarrow}C^{(0)}[\boldsymbol{\gamma}]$, see Fig. 4(a). Verifying this MC against $\varphi' = \mathbb{P}_{\leq 0.3}[\Diamond T \cup \{s_\top\}]$ yields $\mathbb{P}[D^{(0)}, s_0 \models \Diamond T \cup \{s_\top\}] = \boldsymbol{\gamma}(s_0) = 0.2 \leq 0.3$, i.e., the set $C^{(0)}$ is not a CE. We now expand the initial state, i.e., $C^{(1)} = \{s_0\}$ and let $D^{(1)} := \mathcal{D}_{r_0}{\downarrow}C^{(1)}[\boldsymbol{\gamma}]$, see Fig. 4(b). Verifying $D^{(1)}$ yields $\mathbb{P}[D^{(1)}, s_0 \models \Diamond T \cup \{s_\top\}] = 1 \cdot \boldsymbol{\gamma}(s_1) = 0.6 > 0.3$. Thus, the set $C^{(1)}$ is critical

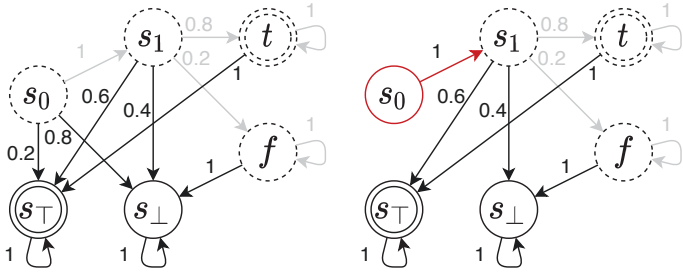

Fig. 4. Finding a CE to $\mathcal{D}_{r_0}$ and φ from Fig. 3 using the rerouting vector $\boldsymbol{\gamma} = lb^{\mathcal{R}}$.

Algorithm 1: Counterexample construction based on rerouting.

Input : An MC $\mathcal{D}_r$ a property $\varphi \equiv \mathbb{P}_{\bowtie\lambda}[\Diamond T]$ s.t. $\mathcal{D}_r \not\models \varphi$, a rerouting vector $\boldsymbol{\gamma}$.
Output : A conflict K for $\mathcal{D}_r$ and φ.

1 $i \leftarrow 0$, $K^{(i)} \leftarrow \emptyset$
2 **while** *true* **do**
3 $\quad C^{(i)}, H^{(i)} \leftarrow \text{reachableViaHoles}(\mathcal{D}_r, K^{(i)})$
4 $\quad D^{(i)} \leftarrow \mathcal{D}_r{\downarrow}C^{(i)}[\boldsymbol{\gamma}]$
5 $\quad$ **if** $\mathbb{P}[D^{(i)} \models \Diamond T \cup \{s_\top\}] \not\bowtie \lambda$ **then return** $K^{(i)}$;
6 $\quad \overline{s} \leftarrow \text{chooseToExpand}(H^{(i)}, K^{(i)})$
7 $\quad K^{(i+1)} = K^{(i)} \cup \text{supp}(\mathcal{B}(\overline{s}))$
8 $\quad i \leftarrow i + 1$
9 **end while**

and the corresponding conflict is $K_{C^{(1)}} = \text{supp}(s_0) = \{X\}$. This is smaller than the naively computed conflict $\{X, Y\}$.

Greedy state expansion strategy Recall from Fig. 2 that for an MC $\mathcal{D}_r$ with $\mathcal{D}_r \not\models \varphi$, multiple CEs may exist inducing different conflicts. An efficient expansion strategy should yield a CE that induces a small amount of relevant parameters (to prune more family members) and this CE is preferably obtained by a small number of model-checking queries. The method presented in Alg. 1 meets these criteria. The algorithm expands multiple states between subsequent model checks, while expanding only states that are associated with parameters that are relevant. In particular, in each iteration, we keep track of the set $K^{(i)}$ of relevant parameters optimistically starting with $K^{(0)} = \emptyset$. We compute (see line 3) the set $C^{(i)}$ of states that are reachable from the initial state via states which are associated only with relevant parameters in $K^{(i)}$, i.e., via states for which $\text{supp}(\mathcal{B}(s)) \subseteq K^{(i)}$. Here, $H^{(i)}$ represents a state exploration 'horizon': the set of states reachable from $C^{(i)}$ but containing some (still) irrelevant parameters. We then construct the corresponding rerouting $D{\downarrow}C^{(i)}[\boldsymbol{\gamma}]$ and check whether it is a CE. Otherwise, we greedily choose a state $\overline{s}$ from the horizon $H^{(i)}$ containing the least number of irrelevant parameters and add these parameters to our

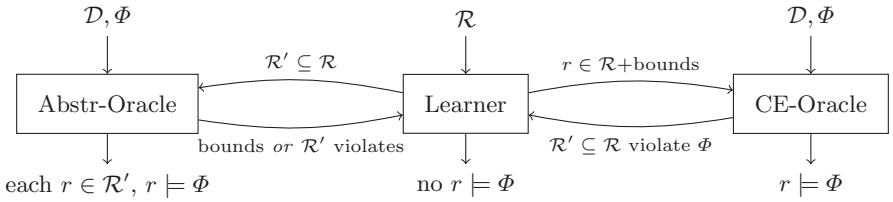

Fig. 5. Conceptual hybrid (dual-oracle) synthesis

.

conflict (see line 7). The resulting conflict may not be minimal, but is computed fast. Our algorithm applies to probabilistic liveness properties[2] too using $\boldsymbol{\gamma} = ub^{\mathcal{R}}$.

Computing bounds We compute $lb^{\mathcal{R}}$ and $ub^{\mathcal{R}}$ using an abstraction [10]. The method considers some set $\mathcal{R}$ of realizations and computes the corresponding *quotient Markov decision process (MDP)* that over-approximates the behavior of all MCs in the family $\mathcal{R}$. Model checking this MDP yields an upper and a lower bound of the induced probabilities for all states over all realizations in $\mathcal{R}$. That is, $\mathit{Bound}(\mathcal{D}, \mathcal{R})$ computes $lb^{\mathcal{R}} \in \mathbb{R}^S$ and $ub^{\mathcal{R}} \in \mathbb{R}^S$ such that for each $s \in S$:

$$lb^{\mathcal{R}}(s) \;\leq\; \min_{r \in \mathcal{R}} \mathbb{P}[\mathcal{D}_r, s \models \Diamond T] \;\leq\; \max_{r \in \mathcal{R}} \mathbb{P}[\mathcal{D}_r, s \models \Diamond T] \;\leq\; ub^{\mathcal{R}}(s).$$

To allow for refinement, two properties are crucial (with point-wise inequalities):

$$1.\ lb^{\mathcal{R}} \leq lb^{\mathcal{R}'} \wedge ub^{\mathcal{R}} \geq ub^{\mathcal{R}'} \text{ for } \mathcal{R}' \subseteq \mathcal{R} \quad \text{and} \quad 2.\ lb^{\{r\}} = ub^{\{r\}} \text{ for } r \in \mathcal{R}.$$

In [10], the abstraction and refinement together define an abstraction-refinement loop (AR) that addresses the feasibility problem. In the worst case, this loop analyses $2 \cdot |\mathcal{R}|$ quotient MDPs, which (as of now) may be arbitrarily larger than the number of family members they represent.

5 Hybrid Dual-Oracle Synthesis

We introduce an extended synthesis loop in which the abstraction-based reasoning is used to prune the family $\mathcal{R}$, and to accelerate the CE-based oracle from Sect. 4. The intuitive idea is outlined in Fig. 5. Note that if the CE-based oracle is not exploited, we emulate AR (explained in computing bounds above), whereas if the abstraction oracle is not used, we emulate CEGIS (with the novel oracle).

Let us motivate combining these oracles in a flexible way. The naive version outlined in the previous section assumed a single abstraction step, and invokes CEGIS with the bounds obtained from that step. Evidently, the better (tighter) the bounds $\boldsymbol{\gamma}$, the better the CEs. However, the abstraction-based bounds for $\mathcal{R}$ may be very loose. These bounds can be improved by splitting the set $\mathcal{R}$ and using the bounds on the two sub-families. The idea is to run a limited number of

[2] Some care is required regarding loops, see [9].

Algorithm 2: Hybrid (dual-oracle) synthesis.

Input : A family $\mathcal{D}$, a reachability property φ.
Output : Either a member r in $\mathcal{D}$ with $r \models \varphi$, or no such r exists in $\mathcal{D}$

1 $\overline{\mathcal{R}} \leftarrow \{\mathcal{R}^{\mathcal{D}}\}$; `// each analysed (sub-)family also holds bounds`
2 $\delta_{CEGIS} \leftarrow 1$; `// time allocation factor for CEGIS`
3 **while** *true* **do**
4 result,$\overline{\mathcal{R}}', \sigma_{AR}, t_{AR} \leftarrow$AR.run$(\overline{\mathcal{R}}, \varphi)$
5 **if** *result*.decided() **then return** *result*;
6 CEGIS.setTimeout$(t_{AR} \cdot \delta_{CEGIS})$
7 $result, \sigma_{CEGIS}, \overline{\mathcal{R}}'' \leftarrow$ CEGIS.run$(\overline{\mathcal{R}}', \varphi)$
8 **if** *result*.decided() **then return** *result*;
9 $\delta_{CEGIS} \leftarrow \sigma_{CEGIS}/\sigma_{AR}$
10 $\overline{\mathcal{R}} \leftarrow \overline{\mathcal{R}}''$
11 **end while**

AR steps and then invoke CEGIS. Our experiments reveal that it can be crucial to be adaptive, i.e., the integrated method must be able to detect at run time when to switch.

The proposed hybrid method switches between AR and CEGIS, where we allow for refining during the AR phase and use the obtained refined bounds during CEGIS. Additionally, we estimate the efficiency σ (e.g., the number of pruned MCs per time unit) of the two methods and allocate more time t to the method with superior performance. That is, if we detect that CEGIS prunes sub-families twice as fast as AR, we double the time in the next round for CEGIS. The resulting algorithm is summarized in Alg. 2. Recall that AR (at line 5) takes one family from $\overline{\mathcal{R}}$, either solves it or splits it and returns the set of undecided families $\overline{\mathcal{R}}'$. In contrast, CEGIS processes multiple families from $\overline{\mathcal{R}}'$ until the timeout and then returns the set of undecided families $\overline{\mathcal{R}}''$. This workflow is motivated by the fact that one iteration of AR (i.e., the involved MDP model-checking) is typically significantly slower that one CEGIS iteration.

Remark 1. Although the developed framework for integrated synthesis has been discussed in the context of feasibility with respect to a single property φ, it can be easily generalized to handle *multiple*-property specifications as well as to treat *optimal* synthesis. Regarding multiple properties, the idea remains the same: Analyzing the quotient MDP with respect to multiple properties yields multiple probability bounds. After initiating a CEGIS-loop and obtaining an unsatisfiable realization, we can construct a separate conflict for each unsatisfied property, while using the corresponding probability bound to enhance the CE generation process. Optimal synthesis is handled similarly to feasibility, but, after obtaining a satisfiable solution, we update the optimizing property to exclude this solution: e.g., for maximal synthesis this translates to increasing the threshold of the maximizing property. Having exhausted the search space of family members, the last obtained solution is declared to be the optimal one.

model	$\|K\|$	$\|\mathcal{R}^{\mathcal{D}}\|$	MDP size	avg. MC size	model	$\|K\|$	$\|\mathcal{R}^{\mathcal{D}}\|$	MDP size	avg. MC size
Grid	8	65k	11.5k	1.2k	*Pole*	17	1.3M	6.6k	5.6k
Maze	20	1M	9k	5.4k	*Herman*	8	0.5k	48k	5.2k
DPM	16	43M	9.5k	2.2k	*Herman**	7	3.1M	6k	1k

Table 1. Summary of the benchmarks and their statistics

6 Experimental evaluation

Implementation. We implemented the hybrid oracle on top of the probabilistic model checker Storm [18]. While the high-performance parts were implemented in C++, we used a python API to flexibly construct the overall synthesis loop. For SMT solving, we used Z3 [29]. The tool chain takes a PRISM [27] or JANI [6] sketch and a set of temporal properties, and returns a satisfying realization, if such exists, or outputs that such realization does not exist. The implementation in the form of an artefact is available at https://zenodo.org/record/4422543.

Set-up. We compare the adaptive oracle-guided synthesis with two state-of-the-art synthesis methods: program-level CEGIS [9] using a MaxSat CE generation [16,41] and AR [10]. These use the same architecture and data structures from Storm. 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 on a single thread. The benchmarks consists of five different models, see Table 1, from various domains that were used in [9,10]. As opposed to the benchmark considered in [9,10], we use larger variants of *Grid* and *Herman* to better demonstrate differences in the performance of individual methods.

To investigate the scalability of the methods, we consider a new variant of the *Herman* model, that allows us to scale the number of randomization strategies and thus the family size. In particular, we will compare performance on two instances of different sizes: *small Herman** (5k members) and *large Herman** (3.1M members, other statistics are reported in Table 1).

To reason about the pruning efficiency of different synthesis methods, we want to avoid feasible synthesis problems, where the order of family exploration can lead to inconsistent performance. Instead, we will primarily focus on non-feasible problems, where all realizations need to be explored in order to prove unsatisfiability. The experimental evaluation is presented in three parts. (1) We evaluate the novel CE construction method and compare it with the MaxSat-based oracle from [9]. (2) We compare the hybrid synthesis loop with the two baselines AR and CEGIS. (3) We consider novel hard synthesis instances (multi-property synthesis, finding optimal programs) on instances of the model *Herman**.

Comparing CE construction methods We consider *the quality of the CEs* and *their generation time*. In particular, we want to investigate (1) whether using CEs-modulo-families yields better CEes, (2) how the quality of CEs from the smart oracle compares to the MaxSat-based oracle, and how their time consumption compares. As a measure of quality of a CE, the average number of its relevant parameters w.r.t. the total number of its parameters is taken. That is, smaller

model		CE quality			performance					
		MaxSat [16]	state expansion (new)		CEGIS [9]		AR [10]		Hybrid (new)	
			trivial	non-trivial	iters	time	iters	time	iters	time
Grid		0.59 (0.025)	0.50 (0.001)	0.50	613	30	5325	486	(285, 11)	**6**
	∗	0.74 (0.026)	0.65 (0.001)	0.65	1801	93	6139	540	(2100, 127)	**33**
Maze		0.21 (0.247)	0.55 (0.009)	0.38	290	5449	49	17	(105, 13)	**7**
	∗	0.24 (2.595)	0.63 (0.012)	0.46	301	6069	63	26	(146, 17)	**9**
DPM		0.32 (0.447)	0.61 (0.007)	0.53	2906	2488	299	25	(631, 143)	**23**
	∗	0.33 (0.525)	0.49 (0.006)	0.42	3172	2782	1215	81	(2374, 545)	**76**
Pole		-	0.87 (0.062)	0.16	-	-	309	12	(3, 5)	**1**
	∗	-	0.54 (0.041)	0.29	-	-	615	23	(80, 61)	**6**
Herman		-	0.91 (0.011)	0.50	-	-	171	86	(24, 1)	**9**
	∗	-	0.88 (0.016)	0.87	-	-	643	269	(485, 13)	**29**

Table 2. CE quality for different methods and performance of three synthesis methods. For each model/property, we report results for two different thresholds where the symbol '∗' marks the one closer to the feasibility threshold, representing the more difficult synthesis problem. Symbol '-' marks a two-hour timeout. **CE quality**: The presented numbers give the CE quality (i.e., the smaller, the better). The numbers in parentheses represent the average run-time of constructing one CE in seconds (run-times for constructing CE using non-trivial bounds are similar as for trivial ones and are thus not reported). **Performance**: for each method, we report the number of iterations (for the hybrid method, the reported values are iterations of the CEGIS and AR oracle, respectively) and the run-time in seconds.

ratios imply better CEs. To measure the influence of using CEs-modulo-families, two types of bounds are used: (i) trivial bounds (i.e., $\boldsymbol{\gamma} = \mathbf{0}$ for safety and $\boldsymbol{\gamma} = \mathbf{1}$ for liveness properties), and (ii) non-trivial bounds corresponding to the entire family $\mathcal{R}^{\mathcal{D}}$ representing the most conservative estimate. The results are reported in (the left part of) Table 2. In the next subsection, we investigate this same benchmark from the point of view of the performance of the synthesis methods, which also shows the immediate effect of the new CE generation strategy.

The first observation is that using non-trivial bounds (as opposed to trivial ones) for the state expansion approach can drastically decrease the conflict size. It turns out that the CEs obtained using the greedy approach are mostly larger than those obtained with the MaxSat method. However (see *Grid*), even for trivial bounds, we may obtain smaller CEs than for MaxSat: computing a minimal-command CE does not necessarily induce an optimal conflict. On the other hand, comparing the run-times in the parentheses, one can see that computing CEs via the greedy state expansion is orders of magnitude faster than computing command-optimal ones using MaxSat. It is good to realize that the greedy method makes at most $|K|$ model-checking queries to compute CEs, while the MaxSat method may make exponentially many such queries. Overall, the greedy method using the non-trivial bounds is able to obtain CEs of comparable quality as the MaxSat method, while being orders of magnitude faster.

Performance comparison with AR/CEGIS We compare the hybrid synthesis loop from Sect. 5 with two state-of-the-art baselines: CEGIS and AR. The results are displayed in (the right half of) Table 2. *In all 10 cases, the hybrid method outperforms the baselines. It is up to an order of magnitude faster.*

Let us discuss the performance of the hybrid method. We classify benchmarks along two dimensions: (1) the performance of CEGIS and (2) the performance of AR. Based on the empirical performance, we classify (*Grid*) as good-for-CEGIS (and not for AR), *Maze*, *Pole* and *DPM* as good-for-AR (and not for CEGIS), and *Herman* as hard (for both). Roughly, AR works well when the quotient MDP does not blow up and its analysis is precise due to consistent schedulers, i.e., when the parameter dependencies are not crucial for a precise analysis. CEGIS performs well when the CEs are small and fast to compute. On the other hand, synthesis problems for which neither pure CEGIS nor pure AR are able to effectively reason about non-trivial subfamilies, inherently profit from a hybrid method. The main point we want to discuss is *how the hybrid method reinforces the strengths of both methods, rather than their weaknesses.*

In the hybrid method, there are two factors that determine the efficiency: (i) *how fast do we get bounds on the reachability probability that are tight enough to enable construction of good counterexamples?* and (ii) *how good are the constructed counterexamples?* The former factor is attributed to the proposed adaptive scheme (see Alg. 2), where the method will prefer AR-like analysis and continue refinement until the computed bounds allow construction of small counterexamples. The latter is reflected above. Let us now discuss how these two aspects are reflected in the benchmarks.

In good-for-CEGIS benchmarks like *Grid*, after analyzing a quotient MDP for the whole family, the hybrid method mostly profits from better CEs yielding better bounds, thus outperforming CEGIS. Indeed, the CEs are found so fast that the bottleneck is no longer their generation. This also explains why the speedup is not immediately translated to the speedup on the overall synthesis loop. In the good-for-AR benchmark *DPM*, the hybrid method provides only a minor improvement as it has to perform a large number of AR-iterations before the novel CE-based pruning can be effectively used. This can be considered as the worst-case scenario for the hybrid method. On other good-for-AR benchmarks like *Maze* and *Pole*, the good performance on AR allows to quickly obtain tight bounds which can then be exploited by CEGIS. Finally, in hard models like *Herman*, abstraction-refinement is very expensive, but even the bounds from the first round yield bounds that, as opposed to the trivial bounds, now enable good CEs: CEGIS can keep using these bounds to quickly prune the state space.

More complicated synthesis problems Our new approach can push the limits of synthesis benchmarks significantly. We illustrate this by considering a new variant of the *Herman* model, *Herman**, and a property imposing an upper bound on the expected number of rounds until stabilization. We put this bound just below the optimal (i.e., the minimal) value, yielding a hard non-feasible problem. The synthesis results are summarized in Table 3. As CEGIS performs poorly on *Herman*, it is excluded here.

synthesis problem	AR iters	AR time	Hybrid iters	Hybrid time
feasibility	81	30s	(274, 1)	**7s**
two properties	97	38s	(274, 1)	**8s**
optimality	531	150s	(571, 7)	**12s**

synthesis problem	AR iters	AR time	Hybrid iters	Hybrid time
feasibility	69k	47h	(14280, 2)	**13.4m**
optimality	83k	55h	(16197, 3)	**16.8m**
5%-optimality	60k	42h	(6421, 7)	**5.1m**

Table 3. The impact of scaling the family size (of the *Herman** model) and handling more complex synthesis problems. The left part shows the results for the smaller variant (5k members), the right part for the larger one (3.1M members).

First, we investigate on *small Herman** how the methods can handle the synthesis for multi-property specifications. We add one feasible property to the (still non-feasible) specification (row 'two properties'). While including more properties typically slows down the AR computation, the performance of the hybrid method is not affected as the corresponding overhead is mitigated by additional pruning opportunities. Second, we consider optimal synthesis for the property as used in the feasibility synthesis. The hybrid method requires only a minor overhead to find an optimal solution compared to checking feasibility. This overhead is significantly larger for AR.

Next, we consider *larger Herman** model having significantly more randomization strategies (3.1M members) that include solutions leading to a considerably faster stabilization. This model is out of reach for existing synthesis approaches: one-by-one enumeration takes more than 27 hours and the AR performs even worse—solving the feasibility and optimality problems requires 47 and 55 hours, respectively. On the other hand, the proposed hybrid method is able to solve these problems within minutes. Finally, we consider a relaxed variant of optimal synthesis (5%-optimality) guaranteeing that the found solution is up to 5% worse than the optimal. Relaxing the optimally criterion speeds up the hybrid synthesis method by about a factor three.

These experiments clearly demonstrate that scaling up the synthesis problem several orders of magnitude renders existing synthesis methods infeasible: they need tens of hours to solve the synthesis problems. Meanwhile, the hybrid method tackles these difficult synthesis problems without significant penalty and is capable of producing a solution within minutes.

7 Conclusion

We present a novel method for the automated synthesis of probabilistic programs. Pairing the counterexample-guided inductive synthesis with the deductive oracle using an MDP abstraction, we develop a synthesis technique enabling faster construction of smaller counterexamples. Evaluating the method on case studies from different domains, we demonstrate that the novel CE construction and the adaptive strategy lead to a significant acceleration of the synthesis process. The proposed method is able to reduce the run-time for challenging problems from days to minutes. In our future work, we plan to investigate counterexamples on the quotient MDPs and improve the abstraction refinement strategy.

References

1. Ábrahám, E., Becker, B., Dehnert, C., Jansen, N., Katoen, J.P., Wimmer, R.: Counterexample generation for discrete-time Markov models: An introductory survey. In: SFM. LNCS, vol. 8483, pp. 65–121. Springer (2014)
2. Alur, R., Bodík, R., Dallal, E., Fisman, D., Garg, P., Juniwal, G., Kress-Gazit, H., Madhusudan, P., Martin, M.M.K., Raghothaman, M., Saha, S., Seshia, S.A., Singh, R., Solar-Lezama, A., Torlak, E., Udupa, A.: Syntax-guided synthesis. In: Dependable Software Systems Engineering, NATO Science for Peace and Security Series, vol. 40, pp. 1–25. IOS Press (2015)
3. Baier, C., de Alfaro, L., Forejt, V., Kwiatkowska, M.: Model checking probabilistic systems. In: Handbook of Model Checking, pp. 963–999. Springer (2018)
4. Baier, C., Hensel, C., Hutschenreiter, L., Junges, S., Katoen, J., Klein, J.: Parametric markov chains: PCTL complexity and fraction-free gaussian elimination. Inf. Comput. **272**, 104504 (2020)
5. Bartocci, E., Grosu, R., Katsaros, P., Ramakrishnan, C.R., Smolka, S.A.: Model repair for probabilistic systems. In: TACAS'11. LNCS, vol. 6605, pp. 326–340 (2011)
6. Bornholt, J., Torlak, E., Grossman, D., Ceze, L.: Optimizing synthesis with metasketches. In: POPL'16. p. 775–788. Association for Computing Machinery (2016)
7. Calinescu, R., Češka, M., Gerasimou, S., Kwiatkowska, M., Paoletti, N.: Efficient synthesis of robust models for stochastic systems. J. of Systems and Softw. **143**, 140–158 (2018)
8. Češka, M., Dannenberg, F., Paoletti, N., Kwiatkowska, M., Brim, L.: Precise parameter synthesis for stochastic biochemical systems. Acta Inf. **54**(6), 589–623 (2017)
9. Češka, M., Hensel, C., Junges, S., Katoen, J.P.: Counterexample-driven synthesis for probabilistic program sketches. In: FM. LNCS, vol. 11800, pp. 101–120. Springer (2019)
10. Češka, M., Jansen, N., Junges, S., Katoen, J.P.: Shepherding hordes of Markov chains. In: TACAS (2). LNCS, vol. 11428, pp. 172–190. Springer (2019)
11. Chatzieleftheriou, G., Katsaros, P.: Abstract model repair for probabilistic systems. Inf. Comput. **259**(1), 142–160 (2018)
12. Chonev, V.: Reachability in augmented interval Markov chains. In: RP'2019. LNCS, vol. 11674, pp. 79–92. Springer (2019)
13. Chrszon, P., Dubslaff, C., Klüppelholz, S., Baier, C.: ProFeat: feature-oriented engineering for family-based probabilistic model checking. Formal Asp. Comput. **30**(1), 45–75 (2018)
14. Classen, A., Cordy, M., Heymans, P., Legay, A., Schobbens, P.Y.: Model checking software product lines with SNIP. Int. J. on Softw. Tools for Technol. Transf. **14**, 589–612 (2012)
15. Daws, C.: Symbolic and parametric model checking of discrete-time Markov chains. In: ICTAC. LNCS, vol. 3407, pp. 280–294. Springer (2004)
16. Dehnert, C., Jansen, N., Wimmer, R., Ábrahám, E., Katoen, J.P.: Fast debugging of PRISM models. In: ATVA. LNCS, vol. 8837, pp. 146–162. Springer (2014)
17. Dehnert, C., Junges, S., Jansen, N., Corzilius, F., Volk, M., Bruintjes, H., Katoen, J.P., Ábrahám, E.: PROPhESY: A PRObabilistic ParamEter SYNnthesis Tool. In: CAV'15. LNCS, vol. 9206, pp. 214–231. Springer (2015)
18. Dehnert, C., Junges, S., Katoen, J.P., Volk, M.: A Storm is coming: A modern probabilistic model checker. In: CAV. LNCS, vol. 10427, pp. 592–600. Springer (2017)

19. Funke, F., Jantsch, S., Baier, C.: Farkas certificates and minimal witnesses for probabilistic reachability constraints. In: TACAS (1). LNCS, vol. 12078, pp. 324–345. Springer (2020)
20. Gerasimou, S., Calinescu, R., Tamburrelli, G.: Synthesis of probabilistic models for quality-of-service software engineering. Autom. Softw. Eng. **25**(4), 785–831 (2018)
21. Ghezzi, C., Sharifloo, A.M.: Model-based verification of quantitative non-functional properties for software product lines. Inf. & Softw. Technol. **55**(3), 508–524 (2013)
22. Hahn, E.M., Hermanns, H., Zhang, L.: Probabilistic reachability for parametric Markov models. Int. J. on Softw. Tools for Technol. Transf. **13**(1), 3–19 (2011)
23. Harman, M., Mansouri, S.A., Zhang, Y.: Search-based software engineering: Trends, techniques and applications. ACM Comp. Surveys **45**(1), 11:1–11:61 (2012)
24. Herman, T.: Probabilistic self-stabilization. Inf. Process. Lett. **35**(2), 63–67 (1990)
25. Jha, S., Gulwani, S., Seshia, S.A., Tiwari, A.: Oracle-guided component-based program synthesis. In: ICSE. p. 215–224. ACM (2010)
26. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic verification of Herman's self-stabilisation algorithm. Formal Aspects of Computing **24**(4), 661–670 (2012)
27. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV. LNCS, vol. 6806, pp. 585–591. Springer (2011)
28. Lanna, A., Castro, T., Alves, V., Rodrigues, G., Schobbens, P.Y., Apel, S.: Feature-family-based reliability analysis of software product lines. Inf. and Softw. Technol. **94**, 59–81 (2018)
29. Lindemann, C.: Performance modelling with deterministic and stochastic Petri nets. SIGMETRICS Perform. Eval. Rev. **26**(2), 3 (1998)
30. Madani, O., Hanks, S., Condon, A.: On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In: AAAI/IAAI. pp. 541–548. AAAI Press / The MIT Press (1999)
31. Martens, A., Koziolek, H., Becker, S., Reussner, R.: Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In: WOSP/SIPEW. pp. 105–116. ACM (2010)
32. Nori, A.V., Ozair, S., Rajamani, S.K., Vijaykeerthy, D.: Efficient synthesis of probabilistic programs. In: PLDI'14. pp. 208–217. ACM (2015)
33. Oliehoek, F.A., Amato, C.: A Concise Introduction to Decentralized POMDPs. Springer Briefs in Intelligent Systems, Springer (2016)
34. Pathak, S., Ábrahám, E., Jansen, N., Tacchella, A., Katoen, J.P.: A greedy approach for the efficient repair of stochastic models. In: NFM'15. LNCS, vol. 9058, pp. 295–309. Springer (2015)
35. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley Series in Probability and Statistics, Wiley (1994)
36. Quatmann, T., Dehnert, C., Jansen, N., Junges, S., Katoen, J.P.: Parameter synthesis for Markov models: Faster than ever. In: ATVA'16. LNCS, vol. 9938, pp. 50–67 (2016)
37. Quatmann, T., Jansen, N., Dehnert, C., Wimmer, R., Ábrahám, E., Katoen, J.P., Becker, B.: Counterexamples for expected rewards. In: FM. pp. 435–452. Springer (2015)
38. Saad, F.A., Cusumano-Towner, M.F., Schaechtle, U., Rinard, M.C., Mansinghka, V.K.: Bayesian synthesis of probabilistic programs for automatic data modeling. Proceedings of the ACM on Programming Languages **3**(POPL), 1–32 (2019)
39. Solar-Lezama, A., Rabbah, R., Bodík, R., Ebcioğlu, K.: Programming by sketching for bit-streaming programs. In: PLDI'05. pp. 281–294. ACM (2005)

40. Vandin, A., ter Beek, M.H., Legay, A., Lluch-Lafuente, A.: Qflan: A tool for the quantitative analysis of highly reconfigurable systems. In: FM. LNCS, vol. 10951, pp. 329–337. Springer (2018)
41. Wimmer, R., Jansen, N., Vorpahl, A., Ábrahám, E., Katoen, J.P., Becker, B.: High-level counterexamples for probabilistic automata. Logical Methods in Computer Science **11**(1) (2015)