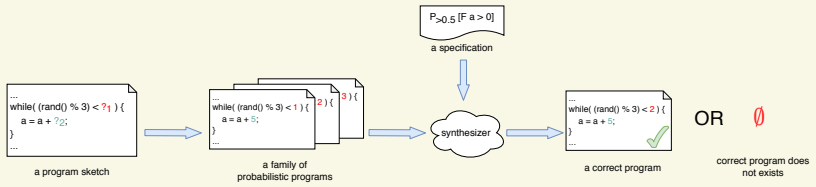
Computer-Aided Synthesis of Probabilistic Models

author: Roman Andriushchenko
supervisor: RNDr. Milan Češka, PhD.

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Motivation

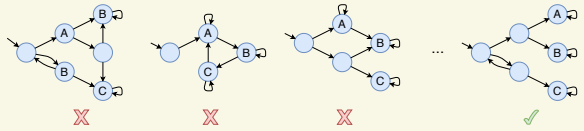
Probabilistic programs serve as an important tool for modelling systems with unpredictable or unreliable behavior, such as communication protocols, controllers for partially observable models, software product lines etc. Designing a system exhibiting a desirable behavior – e.g. a network protocol allowing to increase the packet throughput, or selecting the optimal power management strategy – is a difficult task that involves reasoning over a myriad of alternative designs. To automate this process, we usually start with the so-called **program sketch** [1] – an incomplete description of the program – and let the automatic **synthesizer** fill in this description to obtain a program that satisfies a given **specification**. Essentially, *synthesizer is a program that designs programs.*



To decide whether a candidate program satisfies a specification, we use **Markov chain** as its operational model. One program sketch then corresponds to a **family** of Markov chains, and the goal of an automated synthesizer is to explore this family and identify a chain that satisfies the given specification. Designing such synthesizer represents a tremendous challenge, particularly due to the **double state-space explosion problem**: number of family members as well as the state space of each individual chain grows exponentially wrt. the length of the program.

A 'trivial' example

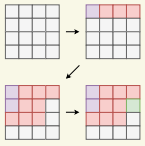
Design an optimal (wrt. to the number of transitions) probabilistic program that uses a fair coin to simulate a random choice between three players.



There are over 3 million candidate designs with at most 6 states. Meanwhile, only 0.001% of them represent optimal solutions. Checking each chain one by one is definitely possible, but how can one *efficiently* identify a satisfying solution?

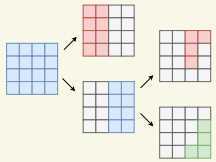
Existing approaches

Counterexample-Guided Inductive Synthesis (CEGIS) [2] analyzes individual chains one by one, but, after encountering an unsatisfying chain (purple), uses a *critical subsystem* of this chain to reject a whole subfamily of chains (red). Critical subsystem – also referred to as *a counterexample* – represents a part of the chain that is sufficient, on its own, to refute the given specification. Thus, any other family member, that shares this subsystem, necessarily refutes the specification as well.



CEGIS

Counterexample-Guided Abstraction Refinement (CEGAR) [3] analyzes all candidate designs at once via *MDP abstraction* – an overapproximation of all the chains in the family. Such analysis produces information about the best-case $p_{\max}$ and the worst-case $p_{\min}$ behavior of the chains in the system. Such information can reveal whether all chains in the family refute the property (red), all chains satisfy the specification (green), or be inconclusive (blue). The latter case indicates that the abstraction is too coarse, and we continue by partitioning the family into refined subfamilies, which are handled analogously.



CEGAR

The proposed solution

Both CEGIS and CEGAR have their advantages and there are families or specifications for which induction, or abstraction, is the preferred approach. Our solution represents a **fusion of both approaches**, combining the power of all-in-one abstraction with the precision of one-by-one induction. The key component for the integration was the following discovery: *we can use information about the best-case $p_{\max}$ and the worst-case $p_{\min}$ behavior of the chains in the family to assist CEGIS in constructing smaller counterexamples.* The intuition behind this reasoning is straightforward: employing global information about the family members relaxes the requirements for the critical subsystems to be the most conservative – we are quite satisfied with a subsystem that is critical only within our family of interest.

The immediate effect of this is evident: smaller counterexamples correspond to larger subfamilies that we are able to reject during one iteration, thus greatly accelerating the computation. The algorithm of the **integrated method** is summarized below:

- *[CEGAR phase]* Analyze family in a CEGAR loop down to a limited depth and collect information about the best-case and the worst-case behavior of the chains in the subfamilies.
- *[CEGIS phase]* For each subfamily, initiate a CEGIS loop, but use the collected data to enhance the counterexample construction algorithm in order to provide smaller critical subsystems.

Experimental evaluation

The proposed method was implemented for *Storm* model checker [4] and was evaluated on numerous practically relevant case studies. The tables below feature selected results, where we report, for each synthesis approach, the synthesis time (in **seconds**) for the two different problems. The values with the asterisk * represent experiments that hit a 10-minute timeout, and therefore the presented values were either roughly approximated from the percentage of processed members, or interpolated from smaller samples.

λ	1-by-1	CEGIS	CEGAR	Integrated
0.003	900*	1.28	<1	<1
0.005	900*	52.74	<1	<1
0.007	950*	750*	<1	<1
0.009	900*	750*	32.14	1.08
0.011	900*	2.1	31.67	1.02
0.013	850*	2.09	31.85	1.01

(a) Problem: design a scheduler for a disk power manager so that the service queue overflows with probability at most λ. Family size: 43M members; size of the MDP abstraction: 27k states; average size of a family member: 4k states.

λ	1-by-1	CEGIS	CEGAR	Integrated
0.74	28k*	1.49	94.86	1.67
0.76	28k*	79.27	10.2	1.63
0.78	28k*	106.13	10.42	4.65
0.80	28k*	82.9	10.33	1.5
0.82	28k*	566.82	10.52	2.27
0.84	28k*	5.75	<1	<1

(b) Problem: devise a policy for a partially observable MDP, such that the agent reaches the target location in a maze with probability at least λ. Family size: 1M members; size of the MDP abstraction: 9k states; average size of a family member: 5k states.

Observe that in all cases the proposed integrated method manages to **significantly outperform state-of-the-art approaches**, sometimes by a margin of **orders of magnitude**. In fact, even when dealing with models for which neither CEGIS nor CEGAR can find a reasonable approach, the integrated method manages to strike **a perfect balance between abstracting and inductive reasoning** in order to efficiently synthesize a program, as illustrated in the experiment below.

λ	1-by-1	CEGIS	CEGAR	Integrated
0.52	78.69	46.01	77.77	2.34
0.54	79.03	45.23	77.53	2.37
0.56	79.54	45.4	79.76	2.3
0.58	78.96	850*	263.23	3.36
0.60	78.08	800*	260.27	3.33
0.62	82.84	800*	276.14	3.31

(a) Problem: use Herman's protocol to self-stabilize a ring after exactly one round with probability at least λ. Family size: 0.5k members; size of the quotient MDP: 54k states; average size of a family member: 2k states.

References

[1] Alur, R. et al. Syntax-guided synthesis. *FMCAD 2013.*
[2] Češka, M. et al. Counterexample-Driven Synthesis for Probabilistic Program Sketches. *Formal Methods – The Next 30 Years, 2019.*
[3] Češka, M. et al. Shepherding Hordes of Markov Chains. *TACAS 2019.*
[4] Hensel, C. et al. The Probabilistic Model Checker Storm. *CoRR abs/2002.07080, 2020.*