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 core subsystem – 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.

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.

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.

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.

References