BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2024
http://excel.fit.vutbr.cz

Using Inheritance Dependencies to Accelerate Abstraction-Based Synthesis of Finite-State Controllers for POMDPs

Aleksandr Shevchenko*

Abstract
Partially observable Markov decision process is an important model for autonomous planning used in many areas, such as robotics and biology. This work focuses on the Abstraction-Refinement framework for the inductive synthesis of finite-state-controllers (FSCs) for POMDPs. The current AR requires model checking of a quotient MDP for an entire set of compatible choices of the subfamily in each iteration. We propose an algorithm, which uses inheritance dependencies to reduce the size of the quotient MDP's mask and to accelerate model checking for subfamilies of FSCs. Depending on the POMDP model, we observe both speedup and slowdown of the overall synthesis. We also introduce a smart version of this algorithm, which preserves all its advantages and reduces its weaknesses. Our approach significantly improves the synthesis of FSCs for POMDPs, in some cases up to the factor 6.

*xshevc01@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Markov chains and Markov decision processes offer a powerful tool for the analysis of stochastic systems and are used in many areas: robotics (planning strategies for robots, error avoidance), biology (population extinction, spread of epidemics), finance (currency market, investment strategies), etc. **Partially observable Markov decision process** (POMDP) is a realistic model, assuming that there is uncertainty about the effects of actions and the true state of the world (Figure 1). The resolution of nondeterminism in POMDP is performed by policies (schedulers).

There are two problems related to POMDP analysis – how to efficiently represent policies and how to find the optimal one. In this work, we focus on the synthesis of **finite-state controllers** (FSCs). They compactly encode policies, using internal memory states (Figure 2). Each FSC keeps the policy size bounded and in most cases, the number of all possible FSCs is infinite. It makes the problem of finding the optimal policy undecidable, but real-world tasks often do not require the optimal solution; one that satisfies the specification is sufficient.

Inductive synthesis of FSCs is a state-of-the-art method introduced in [1], which analyses finite families of FSCs by gradually increasing their memory size. Instead of considering members of the family separately, **abstraction-refinement framework** (AR) for inductive synthesis operates with its abstraction, represeneted by a single **quotient MDP** (Figure 4). The number of refinements (splittings) of the family into subfamilies depends on the bounds (Figure 3), provided by model checking of the quotient MDP. Although the abstraction is common for all subfamilies, model checking is executed for each of them.

In this work, we propose an algorithm, which uses **inheritance dependencies** to accelerate model checking of (sub)families of FSCs for AR (**IDAR**). Model checking results of a family can be also useful for the synthesis of its direct subfamilies (children). Depending on the topology of the POMDP model, this approach is able to reduce the model-checking time on average up to 50%. In some cases, the speedup reaches 6 times.

2. Proposed method

The main idea is to minimise the need of redundant model checkings. Quotient MDP for subfamilies is

just a mask overlaid on the single shared quotient MDP. Model-checking results of the parent for some parts of the abstraction can be preserved for children.

Refinement process of the family reduces the number of available choices by splitting for some parameter (hole). Let the state be **vague**, if there is a chance that its optimal choice from parent's scheduler is not optimal for the child. States, which have choices corresponding to the selected hole, are marked as vague. Predecessors of the vague states are also vague. In vague states, we can not be sure about which choice is optimal even with the parent's scheduler, so we keep all available choices. For non-vague states, there is no need for model checking and they can be omitted. However, implementation of PAYNT [2] and Storm [3] requires at least one action in each state, so we preserve only the optimal choice for non-vague states in the mask.

In the classic version of IDAR (Figure 5), a state is considered vague if its (or some of its successor's) number of available choices decreased after splitting. However, it does not mean that its optimal choice necessarily changed. We developed an extended version to this method – **EIDAR** (Figure 6). Let the choice be **affected** if it is optimal for the parent and either is not available anymore or leads to an affected state (for the child). Unlike IDAR, a state is affected if it certainly lost its optimal choice. All optimal and not available choices are added to a queue $\mathcal{Q}$. Origin states for every choice in $\mathcal{Q}$ are marked as affected. For each affected state s_a, we search through all optimal choices leading to s_a and push then into $\mathcal{Q}$. The algorithm terminates when $\mathcal{Q}$ is empty and we receive a set of affected states. The mask is created similarly to IDAR. Since we mark as affected only certainly affected states, their overall number for EIDAR is less than the number of vague states for IDAR. It results in the smaller mask and in more accelerated model checking.

Since described improvements do not speed up the overall synthesis time for each model, our program can decide whether to use inheritance dependencies for each model or not (**Smart EIDAR**). The main parameters for such decision are:

1. Percentage of vague/affected states.
2. The average number of choices per vague/affected state.
3. The size of the root family.

Switching between IDAR and EIDAR is computationally complex, since we would need to calculate the sets of predecessors for IDAR, leading optimal choices and mapping from choices to states for EIDAR. Therefore, we consider running 20% or at most 100 iterations on EIDAR and then possible switching to classic AR, depending on the described parameters.

3. Experimental results

We have implemented described improvements in PAYNT using Python and C++ binding for efficiency. The implementation was tested on a wide range of models and compared to a classic AR method. The main evaluation criteria were:

1. Similarity of the synthesis result (identical FSCs).
2. The overall synthesis time, time of model building (MB, or **restricting** – applying the mask on the quotient MDP), model checking (MC).
3. Time spent by (E)IDAR.

Experiments (Table 1) shown that on models with the large average number of choices per state, small percentage of affected states and big size of the root family, both IDAR and EIDAR speed up the synthesis better than AR. All models, which were not manually limited by the number of iterations, gave equal optimal FSCs. Experiments also demonstrated that increasing memory size results in better speed up for our methods. A higher number of iterations also improves the speedup, but for slowed down models it may further enhance the deceleration.

SEIDAR operates as follows: starting with EIDAR, if after 20% or at most 100 iterations the percentage of vague states is greater than 90%, or the average number of mask choices per vague state is less than 13, or a common logarithm of the size of the root family is less than 13, we switch to AR.

In refuel-20 without threshold, the synthesis slowed down by 0.78 and 0.75 times for IDAR and EIDAR, respectively. SEIDAR managed to increase these numbers up to 1.32 and speed up the synthesis. Despite the different topology of the models, SEIDAR managed to speed up (or not change) the synthesis in almost all conducted experiments.

4. Conclusions

The result exceeds our expectations, because in addition to model checking, other parts of AR framework are also accelerated. Proposed method significantly contributes to the speedup of the overall synthesis time and improves the scalability of the inductive synthesis.

References

[1] Roman Andriushchenko, Milan Češka, Sebastian Junges, and Joost-Pieter Katoen. Inductive synthesis for probabilistic programs reaches new horizons. In *International Conference on TACAS*, pages 191–209. Springer, 2021.

[2] Roman Andriushchenko, Milan Češka, Sebastian Junges, Joost-Pieter Katoen, and Šimon Stupinský. Paynt: A tool for inductive synthesis of probabilistic programs. In *International Conference on CAV*, volume 12759, pages 856–869, Cham, DE, 2021. Springer Verlag.

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