BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

INDUCTIVE CONTROLLER SYNTHESIS FOR POMDPS WITH RESPECT TO STEADY-STATE PROPERTIES

INDUKTIVNÍ SYNTÉZA KONTROLÉRŮ PRO POMDP VŮČI VLASTNOSTEM VE STABILNÍM STAVU

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **AUTOR PRÁCE**	**ANTONÍN JAROLÍM**
SUPERVISOR **VEDOUCÍ PRÁCE**	**doc. RNDr. MILAN ČEŠKA, Ph.D.**

BRNO 2023

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

148615

Institut: Department of Intelligent Systems (UITS)
Student: **Jarolím Antonín**
Programme: Information Technology
Specialization: Information Technology
Title: **Inductive Controller Synthesis for POMDPs with Respect to Steady-State Properties**
Category: Formal Verification
Academic year: 2022/23

Assignment:

1. Study the state-of-the-art controller synthesis methods for MDPs with the focus on steady-state properties.
2. Design methods allowing for an effective integration of steady-state model checking techniques with the inductive-based controller synthesis.
3. Implement the designed methods in the tool PAYNT.
4. Using suitable benchmarks, perform a detailed experimental evaluation of the implemented methods.

Literature:

- Kochenderfer, M.J., Wheeler, T.A., and Wray K.H, Algorithms for Decision Making, MIT Press 2021.
- Andriushchenko, R., Češka, M., Junges, S., and Katoen, J.P. Inductive synthesis of finite-state controllers for POMDPs. In UAI'22. Proceedings of Machine Learning Research.
- Andriushchenko, R., Češka, M., Junges, S., Katoen, J.P. and Stupinský, Š. PAYNT: A Tool for Inductive Synthesis of Probabilistic Programs. In *CAV 2021*. Springer.
- Velasquez, A. Steady-State Policy Synthesis for Verifiable Control. In *IJCAI* 2019.
- Akshay, S., Bertrand, N., Haddad, S. and Helouet, L. The steady-state control problem for Markov decision processes. In *QEST 2013.* Springer.

Requirements for the semestral defence:
Items 1, 2, and partially 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Češka Milan, doc. RNDr., Ph.D.**
Consultant: Ing. Roman Andriushchenko
Head of Department: Hanáček Petr, doc. Dr. Ing.
Beginning of work: 1.11.2022
Submission deadline: 10.5.2023
Approval date: 11.4.2023

// Abstract
This thesis considers the problem of synthesizing finite-state controllers (FSC) for partially observable Markov decision processes wrt. steady-state properties. The set of candidate FSCs (design-space) is explored using state-of-the-art synthesis methods. The Abstraction-Refinement (AR) method prunes the design-space by considering families of FSCs at once. The novel algorithm generating counter-examples regarding steady-state properties using principles of the counterexample-guided inductive synthesis method is proposed. The experimental evaluation compares the AR method with a one-by-one exploration. It shows that the AR method is faster by orders of magnitude in all but one example, where the low transition rates reduced the speed of the AR method. No other tool is capable of performing such synthesis, so a comparison with other approaches is not available.

Abstrakt
Tato práce se zabývá syntézou konečných automatů pro částečně pozorovatelné Markovovské rozhodovací procesy s ohledem na vlastnosti v ustáleném stavu. Množina přípustných kontrolérů je prozkoumávána pomocí state-of-the-art syntézních metod. Metoda Abstraction-Refinement (AR) prozkoumává tuto množinu tím, že bere v úvahu rodiny kontrolérů najednou. Byl navržen nový algoritmus generující proti-příklady vzhledem ke vlastnostem v ustáleném stavu, pomocí principů metody counterexample-guided inductive synthesis. V experimentální části se porovnává metoda AR se základní one-by-one metodou. Ukáže se, že metoda AR je rychlejší o několik řádů ve většině případů, s výjimkou jednoho, kde nízké hodnoty přechodů snížily její rychlost. Není k dispozici žádný jiný nástroj, který umí provádět takovou syntézu, takže porovnání s jinými přístupy nebylo možné.

Keywords
partially observable Markov decision process, finite state controller synthesis, steady-state properties, family of finite state controllers, abstraction of Markov chains, counter-examples

Klíčová slova
částečně pozorovatelný Markovův rozhodovací proces, syntéza konečných automatů, vlastnosti ve stabilním stavu, rodina konečných automatů, abstrakce rodin Markovských řetězců, protipříklady

Reference
JAROLÍM, Antonín. *Inductive Controller Synthesis for POMDPs with Respect to Steady-State Properties*. Brno, 2023. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. RNDr. Milan Češka, Ph.D.

Rozšířený abstrakt

V reálném světě se často potýkáme s problémy, které obsahují prvek náhody. Tyto problémy se dají modelovat s použitím Markovovských modelů. Ty se vyznačují tím, že přechody mezi stavy jsou dány pravděpodobnostní distribucí a tím, že tato distribuce je závislá pouze na současném stavu. Nejjednodušší Markovův model je Markovův řetezec. Jeho rozšířením o akce vznikne Markovův rozhodovací proces (Markov decision process – MDP), tím se otevře možnost interakce s namodelovaným prostředím pomocí tzv. agenta. Ten vybírá akce na základě stavu, ve kterém se nachází. Akce vybere příslušnou pravděpodobnostní distribuci, která vybere následující stav. Často se však objevují problémy, kde pozice agenta v modelu není přesně známá. Například robot má pouze senzory, které nejsou schopné určit jeho přesnou polohu. Toto nám dovoluje modelovat částečně pozorovatelný Markovovský rozhodovací proces (partially observable MDP – POMDP). V něm agent, místo stavu ve kterém se nachází, dostane pouze omezené informace v podobě *pozorování* (observation), na jehož základě může rozhodnout o další akci. Markovovy procesy jsou často analyzovány, aby se ověřilo, zda-li splňují nějakou vlastnost. Jedna z těchto vlastností definuje pravděpodobnost dosažení nějakého stavu. Další umožňuje přiřadit *odměnu* každému páru stavu a akce, a zkoumat jaké množstí odměny se nasbírá. Poslední vlastnost, která je předmětem této práce, zkoumá jaká je pravděpodobnost, že se agent nachází v daném stavu, když bychom nechali systém bežet nekonečnou dobu. Chování systému se po čase ustálí. Chování agenta v prostředí může být reprezentováno konečným automatem, který v každém stavu (nebo pozorování) vybere jednu akci, a tím vyřeší nedeterminismus.

Tato práce se zabývá syntézou konečných stavových kontrolérů pro částečně pozorovatelné Markovovské rozhodovací procesy s ohledem na vlastnosti v ustáleném stavu. Množina přípustných kontrolérů je prozkoumávána pomocí state-of-the-art syntézních metod. Metoda *abstraction-refinement* (AR) je využita k prozkoumávání této množiny tak, že bere v úvahu rodiny kontrolérů najednou pomocí abstrakce. Tuto abstrakci reprezentuje jeden MDP, jehož analýzou je možné vyřadit celou rodinu najednou. Pokud to není možné, tak je abstrakce *zjemněná* a vzniknou dvě podrodiny, které se následně analyzují. Tento process je pak opakován, dokud není prozkoumán celý prostor přípustných kontrolérů nebo dokud není nalezen kontrolér, který splňuje danou zkoumanou vlastnost. Metoda s opačným přístupem nazývaná *protipříklady řízená induktivní syntéza* (counterexample-guided inductive synthesis, CEGIS), je založena na zkoumání jednoho náhodného kontroléru, který danou vlastnost nespňuje. Cílem je nalézt část kontroléru (a zkoumaného systému), která je dostačující na zavrhnutí tohoto kontroléru. Tato část je následně využita k zavrhnutí větší množiny kontrolérů s podobným chováním. Byl navržen nový algoritmus generující proti-příklady vzhledem ke vlastnostem v ustáleném stavu s využitím principů této metody.

V experimentální části práce se porovnává metoda AR se základní one-by-one metodou, která ověřuje splnitelnost kontroléru jeden po druhém. Je navržena sada experimentů, které mají netriviální vlastnosti ve stabilním stavu. Je ilustrováno řešení některých menších experimentů pomocí poměrně malých kontrolérů. Ukáže se, že metoda AR je rychlejší o několik řádů ve většině případů, s výjimkou jednoho, kde nízké hodnoty přechodů snížily drasticky její rychlost. Tato metoda byla původně implementována pro hledání řešení vzhledem k pravděpodobnostem definujících dosažitelnost stavů. Proto byla provedena analýza modelů také vzhledem k těmto vlastnostem, aby mohla být porovnána její efektivita vzhledem k vlastnosem ve stabilním stavu. Zaznamenané výsledky syntézy v byly pomalejší právě při specifikacích zahrnujících vlastnosti ve stabilním stavu. Metoda je tedy efektivnější v případě vlastnosti dosažitelnosti, ale prokázala svou použitelnost také při syntéze vůči vlastnostem ve stabilním stavu.

Inductive Controller Synthesis for POMDPs with Respect to Steady-State Properties

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of doc. RNDr. Milan Češka, Ph.D. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

....
Antonín Jarolím
May 16, 2023

Acknowledgements

I would like to thank my supervisor, doc. RNDr. Milan Češka, Ph.D., for his essential and critical notes which significantly improved the quality of my thesis and my consultant, Ing. Roman Andriushchenko, for his ideas and valuable comments. Also, I am grateful to my friends and family for their emotional support during my work on a bachelor's thesis. Especially my mother for her unquenchable desire to understand its meaning.

Contents

1 Introduction **2**

2 Preliminaries **5**
2.1 Stochastic modeling using Markov models 5
2.2 Steady-state properties of Markov models 6
2.3 Model checking Markov models 9

3 Inductive Synthesis of Finite-State Controllers for POMDPs **10**
3.1 Families of Deterministic Finite-State Controllers 11
3.2 Abstraction-Refinement Method 14
3.3 Counter-Example Guided Inductive Synthesis Method 17
3.4 PAYNT Tool 19

4 Synthesis of Controllers Regarding Long-Run Average Properties **20**
4.1 Integrating LRA Properties to the AR Method 21
4.2 Generating LRA Property Counter-Example 25

5 Experimental Evaluation **31**
5.1 Benchmark details and models introduction 32
5.2 Impact of used FSC memory for POMDPs with LRA objectives 33
5.3 Evaluating AR performance regarding LRA properties 35
5.4 Investigating limitations of the AR method regarding LRA properties . . . 40

6 Conclusion **45**

Bibliography **47**

Chapter 1

Introduction

Probabilistic systems have a wide range of applications, e.g. robot planning, randomized protocols [5], and analysis of software and hardware systems. The complexity of systems rises very quickly while the robustness requirements are also increasing, as it may have a huge impact on the survival of a company. In the nineties, Intel made a mistake when designing Pentium processors, causing a loss of about 475 million USD. An average car contains dozens of microcontrollers, some responsible for critical safety elements, like vehicle airbag opening crash sensors. Another software flaw caused the death of six cancer patients due to overexposure to radiation in the Therac-25 machine [8]. Therefore, the formal verification of system design is crucial to guarantee the required system properties.

Current state. Formal models allow the modelling of stochastic systems. A Markov chain (MC) is similar to a finite-state machine, except in MC, the successor state is chosen probabilistically. Markov decision process (MDP) extends MC with non-deterministic choices, i.e., the actions. For each permitted action in every state, there is a probabilistic distribution over the successor states [8]. E.g., a robot using actions left, right, etc., to move in its environment in a probabilistic manner. In partially observable MDP (POMDP), a state uncertainty arises. That is, an agent receives observation which has only a probabilistic relationship with the state [14].

Properties of stochastic models are used to verify that a model satisfy given specifications. As the execution of a system is represented by a path [16], the question might be: What is the probability that a certain path is taken? Expected reward property requires each state to have an assigned reward value and express the accumulated expected reward before the given time is reached [5]. Another property, called long-run average reward, is the average reward accumulated per step on every infinite path [7]. That is an elegant way to model performance properties, e.g. power consumption, calculating the number of lost requests [16], the average rate of a particular event, etc. Lastly, long-run average (or so-called steady-state) properties describe the fraction of time spent in a state or a given set of goal states.

Probabilistic model checkers, namely *STORM* [13] and *PRISM* [16], explore a state-space to verify that a model satisfies a given set of specifications. As input, they take a program (formal model) description in the PRISM or JANI language and the set of specifications defined as a conjunction of temporal logic constraints and return 'yes' or 'no', indicating whether the system satisfies the given specifications. In the models with non-determinism, the agent has a wide range of possible behaviors. The objective is to find the behavior, that potentially optimally satisfies the given specifications. A given model

satisfies the property, if there is any behavior that satisfies the property. Finding such behavior in MDP or POMDP is referred to as solving the problem.

Every MDP has only a finite set of possible behaviours, and the set can be enumerated to find the best solution. The solution is guaranteed to be optimal, meaning that no better solution exists. The existence of the optimal solution to POMDPs for long-run average and infinite horizon properties is undecidable [18]. However, the set of possible strategies can be explored to find sub-optimal solution. There are online planning algorithms, which computes the optimal actions during the execution of POMDP. They get a limited amount of time to perform the computation, and the best found action is then played. They can be based on Monte-Carlo tree search [20]. The Monte-Carlo method require the black-box simulator of the POMDPs to the computation of best actions, which is not always possible to supply. On the other hand, they are able to solve larger POMDPs than any other methods. Other POMDP solving methods are based on the belief-curve, but they lack the possibility of verification by formal proofs. Finally, the available policies can be represented as FSCs [9]. In general, such policies are randomized and therefore represented as stochastic FSC (sFSC) [1]. It makes a decision of the following action based on the probability distribution over the actions. If the distribution always selects one action, then the FSC is deterministic. Such FSC have a benefit of explainability and their debugging is easier. Searching for sub-optimal deterministic FSC is implemented in the tool PAYNT.

The *PAYNT* [5] is built on top of *STORM* Python API [13] and allows synthesizing the FSCs for POMDPs [6]. Solving POMDPs requires memory to make better decisions based on the history of actions and observations. The memory is represented as the number of FSC nodes – the larger FSC is able to remember more information. The set of possible FSCs (called *design-space*) increases exponentially when the memory node is added. Therefore, it is obviously unfeasible to do design-space exploration by enumerating all options one by one. Instead, *PAYNT* is pruning design space using more complex methods – abstraction-refinement [10] (AR) and counter-example guided inductive synthesis (CEGIS).

Contribution. The *PAYNT* currently supports the conjunctions of reachability and expected reward properties [5]. The contribution of this thesis is to extend *PAYNT* with an option to specify long-run average properties as program specifications, as well as a formal description of the problem and evaluation of the implemented methods. To provide examples, it would be possible to define constraints like the robot must be in the defined *exploring* state at least half the time, or verify, that the fraction of time spent in the set of error states will not exceed a given threshold for all possible realizations of the system. Another utilization is the maximal (or minimal) synthesis problem, i.e., finding the realization, at which the fraction of time in some state will be as high (or low) as possible.

Related work [22] specifies an algorithm to find a policy maximizing expected reward while satisfying a set of constraints given by steady-state properties. The solution is based on a linear program finding a stochastic policy, which induces Markov chain by applying it to Markov decision process. The steady-state distribution of such Markov chain is computed to validate given constraint. However, the steady-state distribution can be determined only if the underlying Markov chain is recurrent. Therefore, the constraints are added to the linear program finding policies, so the underlying Markov chain is recurrent.

However, my approach is focused on finding policies represented as deterministic FSC for POMDPs. It was proven [11], that sub-optimal POMDP solutions for long-run average properties require only a finite number of memory. Consequentially, the existence problem is decidable. Design-space exploration is done with the utilization of the abstraction-

refinement method implemented in the *PAYNT*. The abstraction aggregates a subset of design-space to one Markov decision process. The analysis of that MDP is used to argue about whole subset, allowing effective exploration. The CEGIS method requires the generation of counter-examples, which is essentially a subset of states. The generation regarding reachability property is based on examining a subset of states while the behavior of other states is changed. Such change, however, drastically changes the behavior of the system when the long-run average properties are concerned. Therefore, such an approach is not feasible and it is shown that it does not work in a model without transient states. The adapted approach is proposed, for generating CEs in the transient states of the model.

Structure of this paper. Chapter 2 presents a necessary theory and introduces a notation used throughout the following chapters. Chapter 3 explains the Abstraction-Refinement and the Counter-Example Guided Inductive Synthesis methods. In the following Chapter 4, the novel algorithm for the generation of Counter-Examples is proposed and the AR method is extended to the synthesis wrt. LRA properties. Chapter 5 presents the detailed experimental evaluation and a comparison with the baseline one-by-one algorithm. Finally, Chapter 6 summarize this thesis and the experimental evaluation.

Chapter 2

Preliminaries

This chapter presents a necessary theory and introduces the notation used throughout the following chapters. First are introduced the Markov models, starting with the Markov chain (MC), the most simple stochastic model without the non-deterministic behavior. The transitions in the MC are based on stochastic distributions over successor states. Extending the Markov chain with the actions creates the possibility of making decisions. Such a model is called the Markov decision process (MDP) and its execution alternates between taking the action and transitioning to the next state based on the selected action. The action is selected based on the current state and the history of actions and visited states. In the partially observable MDP (POMDP), the current state is unclear. Instead of a specific current state, it gets only limited information about the state, called an *observation*. The decision-making in the POMDP is therefore based on the history of actions and observations. The Markov models are often analyzed regarding some properties. One type of such property is reachability, it studies the probability of eventually reaching a given state. Another property studies the behavior of the model in the long run. If one would let the system evolve for an infinite amount of time, it converges to the steady-state. The applications of the mentioned models often have huge timescales, thus, the motivation to study the long-run properties arises. Finally, at the end of the chapter, several tools allowing automatic verification of the models are introduced.

2.1 Stochastic modeling using Markov models

Many real-world situations seem to have a random outcome, but if the problem is examined deeper, then it shows up, that it is actually a complex but deterministic problem. The great examples are for example rolling a dice or tossing a coin. In the modeling of real-world situations, the complex details are often replaced with stochastic behavior. Such models can be formally described using Markov models. The crucial property of Markov models is that the transition values are not based on the history of model execution. That is, the transition distribution over successor states is based solely on the current state. This is the case with all Markov models, hence is this memoryless behavior called *Markov property*. The Markov chain is essentially a probabilistically determined sequence of states.

Definition 1 (MC). [15, 16, 6] Markov chain (MC) is tuple M = (S, s_0, P) where S stands for a set of states and $s_0 \in S$ is the initial state and P is probability distribution function $P(s_i|s_j)$, describing the transition probability to move from state s_j to s_i.

Markov chains are useful for modeling the system with its environment when the behavior of the environment can be described using probabilities. In the case when all probabilities are not clear, the non-deterministic behavior of the agent interacting with the environment arises. Markov decision process (MDP) extends the Markov chain with actions, i.e., the Markov chain is MDP with only one action [6].

Definition 2 (MDP). [8, 22] MDP is tuple M = (S, s_0, Act, P) where Act is a set of actions that can be taken in each state. P: $S \times Act \times S \rightarrow [0, 1]$ is the transition probability function $P(s' \mid s,\ a_k)$, where $a_k \in Act$ and $s, s' \in S$, means the probability of moving from state s to s' when the action a_k is taken. Naturally, the sum of probabilities to move from state $s \in S$ to s', when action $a \in Act$ is taken must be equal to 1:

Markov decision process permits modeling of the non-determinism while the probabilistic choices from MCs are preserved [9]. The agent is interacting with the environment by making decisions. E.g., imagine the environment given by states arranged to the grid. The agent is using the actions left, right, up, and down to move to different states. However, the actions have unclear consequences, the agent may take the action up but accidentally transition to the grid on the left. This uncertainty is modeled using the stochastic distributions over the successor states. E.g., taking action down in the concrete state may lead to bottom, left and right states with the probability of 0.8, 0,1, and 0.1 respectively.

However, many real-world applications of the MDPs will come across the problem, that the state of the system is not perfectly observable. For example, the sensors of the robot are not perfect and there are not able to perfectly determine the state of the robot. However, they provide some information about the state and the agent takes a decision based on them. The special case, when a single observation is obtained in all states is called *blind MDP*. Otherwise, the environment can be described using the POMDP model.

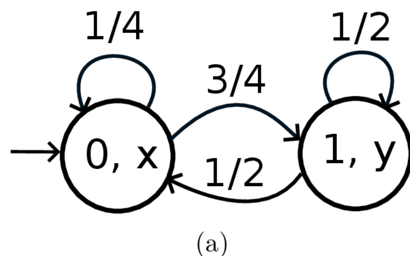
Definition 3 (POMDP). [6, 9] Partially observable MPD (POMDP) extends MDP with observations. POMDP $\mathcal{M} = (S, s_0, Act, P, Z, O)$ where Z is a finite set of observations and O is the observation function, that returns for every state s an observation $Z(s) = z \in Z$.

POMDPs have various applications for example computational biology, robotics [17], image processing and many others. However, the state uncertainty makes the verification of the POMDPs more difficult.

2.2 Steady-state properties of Markov models

If you let the system evolve for an infinite amount of time, the steady-state distribution is the fraction of time spent in each state. Let's assume the procedure of endless repetition of rolling a fair die. If the dice is rolled a few times, it may appear that some values on the dice are more common. However, after the infinite amount of rollings, it would be clear that the probabilities to get a certain value are the same. Such a procedure can be modeled as MC and the steady-state distribution of this model would result in the same probabilities as the described experiment. In general, the steady-state of the system describes the behavior of the model in the long run. The steady-state distribution can be computed only for the MCs. It is not possible for the MDPs and POMDPs, because it is unclear how to solve the non-determinism and thus, the Long-Run Average (LRA) property will be defined.

The steady-state distribution is described first, using the transient probability distribution. Transient probability distribution $\theta_n(s, t)$ is a vector indicating the probability of



(a)

n	$\theta_n(x,x)$	$\theta_n(x,y)$
0	1.00000	0.00000
1	0.25000	0.75000
2	0.43750	0.56250
3	0.39062	0.60938
4	0.40234	0.59766
5	0.39941	0.60059
6	0.40015	0.59985
7	0.39996	0.60004
8	0.40001	0.59999
9	0.40000	0.60000

(b)

Figure 2.1: Example of Markov chain (a) with two states, where x is the initial state and its transient probability distribution (b), which converges to a steady-state.

being in some state $t \in S$ after $n \geq 0$ steps when starting in state $s \in S$ [8, p. 828]. The distribution $\theta_0(s,t)$ is the initial distribution over the initial states. In the case of a single initial state, the probability to be in the initial state is 1. For $n \geq 1$, the vector θ_n is calculated by equation 2.1.

$$\theta_n(s,t) \; = \theta_{n-1}(s,t) \cdot P \tag{2.1}$$

Example 1. Consider a MC M outlined in the Figure 2.1. The transition matrix of M is:

$$P = \begin{bmatrix} 1/4 & 3/4 \\ 1/2 & 1/2 \end{bmatrix}$$

The state x is the initial state, therefore the vector $k_0 = [10]$. Table 2.1b shows the transient probability distribution vectors for $0 < k < 10$ calculated using equation 2.1. The transient probability distribution converges to the steady-state distribution, meaning that the next enumeration will not change the values. Therefore, the steady-state distribution for the MC M is [0.4 0.6].

This enumerating approach of the Equation 2.1 is simple as it requires only a multiplication of a vector by a matrix. However, in the complicated MCs, it takes a very long time to converge [21], or it may never converge. Instead of enumerating the equation, the steady-state distribution $\Pr^\infty$ can be obtained by taking the equation to the limit, that is, $\lim n \to \infty k_n = \Pr^\infty$. This limit exists only in the case when at least one state is absorbing. The state is absorbing when it contains a self-loop. Otherwise, the transient probability distribution will alternate between a finite set of numbers and never converge to any value.

Another approach to obtain steady-state distribution is to solve the following set of linear equations [21].

Definition 4 (Steady-State Distribution). [22] Given a Markov chain $\mathcal{M} = (S, s_0, T)$, the steady-state distribution $\Pr^\infty : S \mapsto [0,1], \sum_{s \in S} \Pr^\infty(s) = 1$, also known as the stationary or invariant distributions, over the state space denotes the proportion of time spent in each state as the number of transitions within $\mathcal{M}$ approaches ∞. This distribution is given by the solution to the system of equations given by 2.2 and 2.3.

$$X * P = X \tag{2.2}$$

Where X is a vector of all states, i.e. $X = [x_1 \; x_2 \; ... \; x_n]$ and P is the transition matrix of the given MC. The normalization equation has to be added to ensure a unique solution.

$$\sum_{s \in S} s = 1 \tag{2.3}$$

Considering MC from example 2.1 it is:

$$[x \; y] * \begin{bmatrix} 1/4 & 3/4 \\ 1/2 & 1/2 \end{bmatrix} = [x \; y]$$
$$x + y = 1$$

Which can also be written as:

$$\begin{aligned} 1/4 * x + 1/2 * y =& x \\ 3/4 * x + 1/2 * y =& y \\ x + y =& 1 \end{aligned}$$

Solving this set of linear equations will produce the same result as that to which 2.1b converged:

$$\Pr^{\infty} = [x \; y] = [0.4 \; 0.6]$$

The steady-state distribution is a very accurate specification of the system, however, when the state-space is large, it becomes difficult to derive anything from it. The LRA property is the sum of steady-state probabilities to be in a given subset of states. This allows to study more general system properties.

Definition 5 (LRA). Let the M be a finite MC, a set $T \subseteq S$ of target states, and $\Pr^{\infty}$ the steady-state vector of MC M. The *LRA* $[T]$ of the MC M is the sum of steady-state values of each target state:

$$LRA\,[T] = \sum_{s \in T} t_{\infty}(s)$$

Critically, the steady-state distribution exists only for Markov chains, because it is unclear how to solve nondeterminism. In other words, the behavior of the agent may change over time and the behavior will never converge to steady-state. The scheduler resolves the non-determinism in MDP (and POMDP) by selecting actions on any state of any path.

Definition 6 (Scheduler). [10] A scheduler for an MDP $M = (S, s_0, Act, \mathcal{P})$ is a function $\sigma : \text{Paths}^M_{fin} \to \text{Act}$ such that $\sigma(\pi) \in \text{Act}(\text{last}(\pi))$ for all $\pi \in \text{Paths}^M_{fin}$. Scheduler σ is memoryless if $\text{last}\,(\pi) = \text{last}\,(\pi') \implies \sigma\,(\pi) = \sigma\,(\pi')$ for all $\pi, \pi' \in \text{Paths}^M_{fin}$. The set of all schedulers of M is Σ^M.

The scheduler defines the behavior of the agent and remains fixed over time. In each state, the one selected action will always be played, thus, the other actions can be omitted. The MDP with only one action in each state is recognized as MC, denoted as M^{σ}. Like that, the steady-state distribution of MC created by every possible scheduler can be obtained and the LRA can be computed based on it. The schedulers $\sigma_{\min}, \sigma_{\max}$ denote the schedulers with minimal and maximal found LRA values.

Definition 7 (LRA$_{\updownarrow}$). Let M be a MDP, a set $\updownarrow = \{min, max\}$ and the $\sigma_{\min}, \sigma_{\max} \in \Sigma^M$ be a schedulers such that $\forall \sigma \in \Sigma^M : LRA\ M^{\sigma_{\min}} \leq LRA\ M^{\sigma} \leq LRA\ M^{\sigma_{\max}}$. The LRA_{min} $= LRA\ M^{\sigma_{\min}}$ and $LRA_{max} = LRA\ M^{\sigma_{\max}}$.

For POMDPs, the number of schedulers is infinite. Therefore, the schedulers $\sigma_{\min}$ and $\sigma_{\max}$ cannot be determined [11, 18].

2.3 Model checking Markov models

Formal probabilistic models are great for modeling probabilistic systems and the objective is often verification, that a model satisfies a specification. The probabilistic programs, such as *PRISM*[1] and *STORM*[2], make the verification automatic. As input, they take the system specification (Markov model) in the *PRISM* or *JANI* language and a set of investigated properties. The finite-horizon properties are limited to a number of steps. Infinite-horizon objectives consider all infinite paths. Some of the many available properties include reachability, reward, and long-run average properties. The reachability properties are denoted as P, they study the probability to reach a given set of target states. It can be limited on the number of steps, e.g., is the given property satisfied in 100 steps? The indefinite reachability is also supported – what is the probability of eventually reaching a given subset of states? The reward assigns each state-action pair a real value and the finite property is the average accumulated reward. The indefinite reward property is called mean payoff or long-run average reward and it is the average reward per step when simulating the MDP [15]. The last type is the long-run average (LRA) property, which studies the average probability to be in a given state.

The specified tools provide an interface to check the feasibility of a property or get the exact value. The feasibility check evaluates whether a certain property holds. E.g., is the probability to reach a target state at least 90%? Such property is denoted as $P_{>0.9}[\,T\,]$. The exact value can be obtained with a check denoted as $P =?\,[\,T\,]$. On models with non-determinism, it is not possible to obtain the exact value, but rather only the minimal and maximal values obtained by the best and worst schedulers. Both model checkers supports model-checking reachability properties on all models and LRA properties on MCs. The PRISM is not able to provide the $LRA_{\updownarrow}$ values on MDP, therefore the STORM API is used. The STORM computes the long-run average properties (Definition7) using value iteration [7], or strategy iteration [15]. Strategy iteration is a dynamic programming technique, which starts with an arbitrary strategy and iteratively improves it until the optimal solution is found. Neither tool is able to perform model-checking regarding LRA properties on POMDPs, because the min/max scheduler existence problem is undecidable. In the next chapter, the eps-optimal solutions for POMDPs are explained.

[1]Prism model checker is available at: https://www.prismmodelchecker.org/
[2]Storm model checker available at: https://www.stormchecker.org

Chapter 3

Inductive Synthesis of Finite-State Controllers for POMDPs

In most real-world control systems, the state of the agent is not clear, due to modeling of inaccuracy, sensing limitations, and so forth. The POMDP is therefore viable model with a wide range of applications in many different fields. The analysis of the POMDP is often focused on finding a policy satisfying some constraints. The constraints specify the desired behavior of the system – reaching a given state of the system, avoidance of fatal states, reaching a subset of states infinitely often, visiting a given set of states in arbitrary order, etc. For a given POMDP and a threshold specifying the probability of success, the existence of the policy is undecidable [18].

The policies generally use memory to obtain a better estimate of the belief of the agent. The memory essentially allows the agent to make better decisions based on the history of observations and actions. Restricting the size of the memory creates a finite set of policies in which the (sub)optimal policy can be found. The policies for POMDPs are commonly represented by policy trees, belief states and FSCs [9]. Point-based value iteration [19] and Monte-Carlo tree search [20] algorithms are superb in finding optimal policies based on belief space. Another approach is finding an optimal policy in a set of either stochastic [1] or deterministic FSCs [9], where the FSC size indicates how much memory is being used. Deterministic FSC is the special case of a stochastic FSC and has advantages in terms of explainability and reproducibility. The current state-of-the-art approach focused on finding optimal deterministic FSC [6] is using oracle-guided inductive synthesis framework which will be explained in this chapter.

The framework can be described as a learner-teacher algorithm. The learner constructs a set of finitely many FSCs (called *design-space*), describing possible realizations of the controller. The teacher explores the design-space to determine which one is the *best* and provides additional information. The exploration may be done by naive enumeration of each FSC. However, the inductive synthesis methods are able to prune a larger subset of FSC at once, allowing much faster exploration. The learner either accepts the proposed FSC as the result of the synthesis or modifies the design-space [6]. The framework supports two types of properties – indefinite-horizon reachability and expected reward. This thesis focuses on extending the framework with long-run average (LRA) properties (see the following chapter).

Problem statement. Given a POMDP $\mathcal{M}$ and a *specification* of the synthesized FSC given by a set of constraints ϕ and at most one optimization objective $\boldsymbol{o}$, find a controller $\mathcal{F}$ satisfying all constraints ϕ and then incrementally improve the FSC $\mathcal{F}$ wrt. optimization objective. Each constraint is limiting a given property by either an upper or lower bound. For an indefinite-horizon reachability property, a constraint is defined as threshold $\lambda \in [0, 1]$, specifying the probability of eventually reaching a set of target states $T \subseteq S$. Let the set of operators $\bowtie \in \{<, >\}$, the FSC is admissible for POMDP $\mathcal{M}$ if the induced MC satisfies $P_{\bowtie\lambda}$. Constraints are defined similarly for expected reward properties with the threshold $\lambda \in \mathbb{R}$. Similarly, the optimization objective is to find FSC minimizing or maximizing a given property. Let the set $\updownarrow = \{min, max\}$, the optimization property is denoted as $P_{\updownarrow}$ or $R_{\updownarrow}$, for reachability and reward property, respectively [6].

The first section explains the formal model for a set of FSCs – a *family* of FSCs and introduces the algorithm which implements the learner. Then, the baseline one-by-one algorithm exploring a given family of FSC will be introduced. The following sections explain the current state-of-the-art inductive synthesis methods. The abstraction-refinement method prunes the design-space by creating an abstraction over set of FSCs [6, 10]. The counter-example guided inductive synthesis method checks whether candidate FSC satisfies specifications and provides CE if not. The CE generalizes to subset of unsatisfying FSC, which can be safely pruned.

3.1 Families of Deterministic Finite-State Controllers

The strategy (also called policy or scheduler) defines the rules deciding the action to play based on past actions and observations. If the strategy is limited in the amount of information it can store, then it is called a finite-memory strategy. If the strategy is finite-memory, then it can be described with finite-state controller [11]. The deterministic FSC represents the strategy and has the following structure.

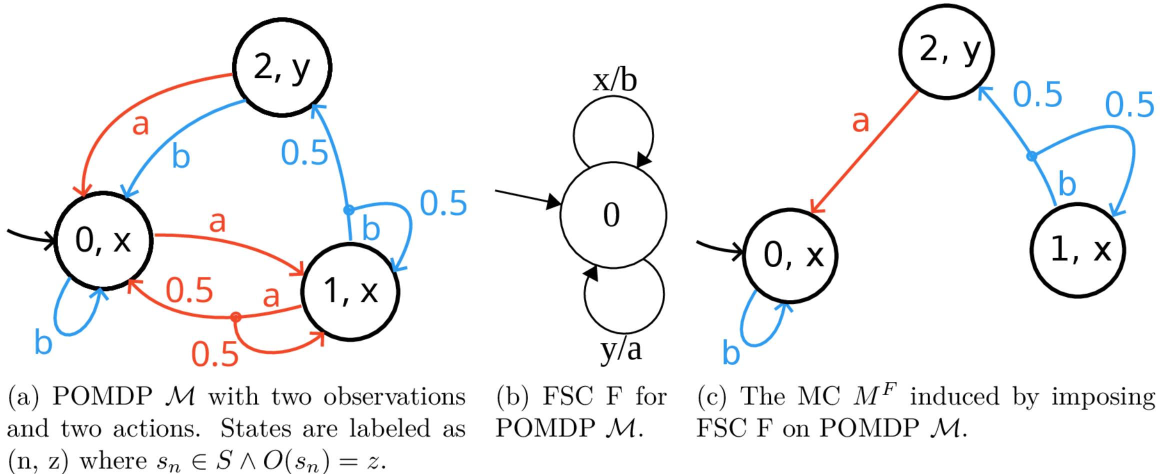
Definition 8 (FSC). [6] Finite-state controller (FSC) for a POMDP $\mathcal{M}$ is a tuple $F = (N, n_0, \gamma, \delta)$, where N is a finite set of nodes, $n_0 \in N$ is the initial node, $\gamma(n, z)$ determines the action when the agent is in node n and observes z, while δ updates the memory node to $\delta(n, z)$, when being in n and observing z. For $|N| = k$, we call an FSC a k-FSC.

The nodes in FSC are also called *internal memory states* [9] and they represent the number of memory that can the agent utilize to better decision-making. Internal memory is changing based on the history of observations. The function δ updates the internal state based on the current state and observation. The function γ resolves the non-determinism by selecting which action will be played at each state and observation combination. Therefore, imposing FSC F on POMDP $\mathcal{M}$ produces MC M^F.

Definition 9 (MC induced by FSC). [6] Imposing k-FSC F onto POMDP $\mathcal{M}$ yields the Markov chain (MC) $M^F = \left(S^F, (s_0, n_0), P^F\right)$ with $S^F = S \times N$ and using $z = O(s)$:

$$P^F\left((s', n') \mid (s, n)\right) = \begin{cases} P\left(s' \mid s, \gamma(n, z)\right) & if\ \delta(n, z) = n' \\ 0 & otherwise. \end{cases}$$

For each POMDP state $s \in S$, there are $|N|$ memory nodes, and therefore, in the resulting M^F there are $|S| \cdot |N|$ states in total. The function P^F maps the corresponding transition function to the internal state selected by the δ. The set of FSC with k memory nodes for a POMDP $\mathcal{M}$ is called family F_k.



(a) POMDP $\mathcal{M}$ with two observations and two actions. States are labeled as (n, z) where $s_n \in S \wedge O(s_n) = z$.

(b) FSC F for POMDP $\mathcal{M}$.

(c) The MC M^F induced by imposing FSC F on POMDP $\mathcal{M}$.

Figure 3.1: Imposing FSC F on POMDP $\mathcal{M}$ yields a Markov chain.

Definition 10 (Family of FSC $\mathcal{F}_k$). [6] A family of full k-FSCs is a tuple $\mathcal{F}_k = (N, n_0, K)$, where N is the set of k nodes, $n_0 \in N$ is the initial node, $K = N \times Z$ is a finite set of parameters each with domain $V_{(n,z)} \subseteq \text{Act} \times N$.

Selecting a value for each parameter from the corresponding domain creates a concrete member of the family. The FSC and POMDP yields MC, therefore, the POMDP $\mathcal{M}$ and family of FSC $\mathcal{F}_k$ naturally yields a family of Markov chains. The teacher explores the family of MCs to select the optimal MC. Remark that the MC is optimal only in the given family because it is always possible to add another memory node, which may improve the optimal value. The family is only a subset of an infinite number of FSC. Therefore, the optimal MC in a given family is called sub-optimal in general.

Example 2. For the POMDP $\mathcal{M}$ depicted in Figure 3.1a, the family of memoryless controllers $\mathcal{F}_1 = (N, n_0, K)$, where $N = \{n_0\}$, $K = \{k_1, k_2\}$ where parameters $k_1 = (n_0, x)$, $k_2 = (n_0, y)$ and both have the same domain $V = \{(a, 0), (b, 0)\}$. With $|V| = |K| = 2$, there are two options for each of the two parameters. Therefore, the design-space describes 4 1-FSCs, and one of them, FSC F is outlined in Figure 3.1b. It is obtained by selecting value $(b, 0)$ and $(a, 0)$ for parameter k_1 and k_2, respectively. FSC F decides when to take which action in POMDP $\mathcal{M}$ yielding MC M_F represented in Figure 3.1c.

A given state s is perfectly observable in POMDP $\mathcal{M}$ if $\forall\ s' \in S \wedge s \neq s'$ is the fact that $O(s) \neq O(s')$, meaning that the observation is unique for state s. For many POMDP problems, the memory is not required in perfectly observable states, but in a family of FSC (Definition 10), there are k nodes for each observation z. Therefore, there is a model of a reduced family given by a memory model $\mu : Z \rightarrow \mathbb{N}$, reducing the number of memory nodes used in observation $z \in Z$ to $\mu(z)$. This reduces the size of parameter domains, significantly reducing the size of the design-space. Another benefit is that memory needed to store and execute the controller is also reduced, which is useful in resource-aware applications [6]. The model of a reduced family is defined as follows:

Definition 11 (Reduced family of FSC $\mathcal{F}_\mu$). [6] A reduced family $\mathcal{F}_\mu$ given by the memory model μ is a sub-family of $\mathcal{F}_k$. The number of nodes $|N| = \max_{z \in Z}\{\mu(z)\}$, $(n, z) \in K$ implies $n \leq \mu(z)$, and the domains $V_{(n,z)}$ are as in $\mathcal{F}_k$. If the memory update function

updates δ(n, z) = n' and n' > μ(z'), then the memory update is invalid and update δ(n, z) = n_0 is used.

In the described framework, the learner creates a finite design-space by restricting searched FSCs by a number of nodes. It is beneficial to search in small FSCs first because 1) small FSCs are well explainable and 2) an admissible controller is often found fast as some POMDPs require only a few memory nodes. Therefore, the strategy is to search memoryless (1-FSCs) controllers first and incrementally add memory. The Algorithm 1 constructs the family of a given size in each iteration. Then, reduces memory used in perfectly observable states using memory model μ. After that, the inner exploration loop (teacher) analyzes constructed family and returns found satisfying controller, or `NONE`. If the controller satisfying set of constraints is not found in family $\mathcal{F}_k$, then the family $\mathcal{F}_{k+1}$ is explored. If the controller is found and the optimization property is not specified, then the satisfying controller is the result of the synthesis. Whenever is the optimization property present, there is no stopping criterion specified. However, user can stop the synthesis loop whenever he wants or it can stop after a specified time.

Algorithm 1 Learner loop

Input: POMDP $\mathcal{M}$, set of constraint ϕ, optimization property $\boldsymbol{o}$
Output: Best FSC F^* or `UNSAT`

```
1: F* ← NONE
2: k ← 1
3: while true do
4:     F_k ← constructFamily(M)                  ▷ Definition 10
5:     F_μ ← reduceFamily(F_k, M)                ▷ Definition 11
6:     F* ← exploreFamily(M, F_μ, φ, o)          ▷ Initiate inner exploration loop
7:     if F* is not NONE and o is NONE then
8:         return F*
9:     end if
10:    k ← k + 1
11: end while
```

3.1.1 Baseline one-by-one family exploration algorithm

The teacher is responsible for exploring a design-space, naturally, the straightforward method is reasoning about each family member one by one. The Algorithm 2 iterates over each FSC F in a family $\mathcal{F}$, imposes F onto POMDP $\mathcal{M}$ yielding MC M_F and checks the feasibility of constraints ϕ. If the MC is feasible and improves so far best found FSC F*, then the FSC is saved to be returned at the end. The MC improves best FSC F*, if the FSC F* is `NONE` or when the value of the property under F is lower (greater) than MC under FSC F* for a given safety (liveliness) optimization property.

The one-by-one exploration is understandably unfeasible for large synthesis problems. Therefore, there are better techniques based on reasoning about families of MCs.

Algorithm 2 One-by-one algorithm

Input: POMDP $\mathcal{M}$, family of k-FSC $\mathcal{F}_\mu$, set of constraint ϕ, optimization property $\boldsymbol{o}$
Output: Best satisfying FSC F^* or `NONE`

```
1: F* ← NONE
2: while F ≠ ∅ do
3:     F ← any(F)
4:     F ← F \ {F}
5:     M^F ← M under F                                  ▷ Definition 9
6:     if ∀φ ∈ ϕ : M^F ⊨ φ then
7:         if M^F improves F* wrt. o then
8:             F* ← F
9:         end if
10:    end if
11: end while
12: return F*
```

3.2 Abstraction-Refinement Method

The abstraction-refinement method is based on considering sets of MCs at once. The formal model for a set of MCs is called the family of MCs. The stochastic model called *quotient MDP* allows to the creation of abstraction over a given family. This allows for verification of the behavior of the entire family by one model-check call, but the verification result is over-approximation. Model-checking the abstraction MDP wrt. a given property φ provides interesting results, allowing to discard the entire family. If the abstraction is too coarse then it is necessary to refine the family. The refinement is essentially splitting the family of MCs into two subsets of MCs. After splitting, the new abstraction over each subset is created and the process is recursively repeated. This concept will be described in detail.

Definition 12 (Family of MCs). [10] *A family of MCs is a tuple* $\mathcal{D} = (S, s_0, K, \mathcal{B})$ where S is a finite set of states and $s_0 \in S$ is an initial state, K is a finite set of discrete parameters with domains $V_k \subseteq S$ for each $k \in K$, and $\mathcal{B} : S \to \text{Distr}(K)$ is a family of transition probability matrices.

The function $\mathcal{B}$ maps states to distribution over parameters [10]. Selecting a value for each parameter from the corresponding domain yields a concrete MC called the realization of the family.

Definition 13 (Realization). [10, 4] 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 $k \in K$. Realization r induces MC $\mathcal{D}_r = (S, s_0, \mathcal{B}_r)$ where $\mathcal{B}_r$ is the transition probability matrix in which each $k \in K$ in P is replaced by r(k). The set of all realizations of $\mathcal{D}$ is denoted as $\mathcal{R}^{\mathcal{D}}$.

Subset of all realizations $\mathcal{R}^{\mathcal{D}}$ is called a sub-family. The number of realizations in family $\mathcal{F}$ is exponential in the number of parameters. Enumerating realizations [10] is technically the same approach as Alg. 2, because it requires model checking each MC one-by-one.

Example 3. The family of MCs obtained by imposing family of 1-FSCs $\mathcal{F}_1$ on POMDP $\mathcal{M}$ outlined in Figure 3.1a is $\mathcal{D} = \left(S^{\mathcal{D}}, s_0, K, \mathcal{B}\right)$, where $S^{\mathcal{D}} = S$, the set of parameters K=$\{k_0, k_1, k_2\}$, with domains $V_{k_0} = \{s_0, s_1\}$, $V_{k_1} = \{s_0, s_2\}$, $V_{k_2} = \{s_1\}$, $V_{k_3} = \{s_0\}$ and

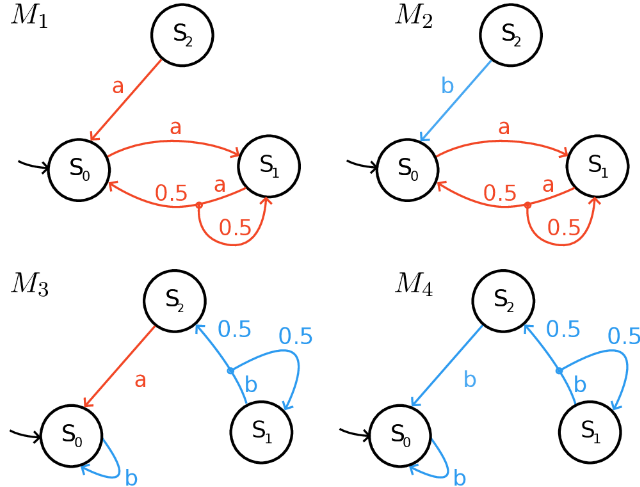


Figure 3.2: Set of Markov chains given by enumerating realizations of the family $\mathcal{D}$.

the family of transition probability matrices $\mathcal{B}$ is defined by:

$$\mathcal{B}(s_0) = 1 : k_0,$$

$$\mathcal{B}(s_1) = 0.5 : k_1 + 0.5 : k_2,$$

$$\mathcal{B}(s_2) = 1 : k_3.$$

Concrete realization $r_1 \in \mathcal{R}^{\mathcal{D}}$ arises by selecting values s_1 and s_2 for parameters k_0 and k_1, respectively and the only available values for parameters k_2 and k_3. The realization r_1 and all other realizations from $\mathcal{R}^{\mathcal{D}}$ are in Figure 3.2.

Instead of enumerating realizations to decide the feasibility of each realization separately, there is a stochastic model in which all realizations are possible at once. It is achieved by allowing to switch from one realization to another mid-execution [3]. Switching realizations changes the probability distributions over the successor states. This concept is identical to using actions and therefore, the described model is conclusively MDP.

Definition 14 (Quotient MDP). [3, 10] Let $\mathcal{D} = (S, s_0, K, \mathcal{B})$ be a family of MCs. A quotient MDP of $\mathcal{D}$ is an MDP $M^{\mathcal{D}} = \big(S, s_0, \mathcal{R}^{\mathcal{D}}, \mathcal{P}\big)$, where $\mathcal{P}(\cdot)(r) \equiv \mathcal{B}_r$. The restriction of $M^{\mathcal{D}}$ wrt. set of realizations $\mathcal{R} \subseteq \mathcal{R}^{\mathcal{D}}$ is the MDP $M^{\mathcal{D}}[\mathcal{R}] = \big(S, s_0, \mathcal{R}^{\mathcal{D}[\mathcal{R}]}, \mathcal{P}\big)$ where $\mathcal{R}^{\mathcal{D}[\mathcal{R}]} = \{a_r | r \in \mathcal{R}\}$.

The quotient MDP is able to execute the path of every realization of the family because the actions at each state correspond to any realization [10]. However, this allows for the execution of paths, which do not correspond to the behavior of any family member. This is called over-approximation and it is demonstrated in the following example.

Example 4. The quotient MDP $M^{\mathcal{D}}$ of the family $\mathcal{D}$ from Example 3 allows to switch between 4 realizations of the family after each probabilistic transition to the successor state. It is possible to always select one realization – execute $s \xrightarrow{r_*} s' \xrightarrow{r_*} \ldots \xrightarrow{r_*} s^{\infty}$ where $r_* \in \mathcal{R}^{\mathcal{D}}$. Notice (see Figure 3.2) that in neither MC of the family $\mathcal{D}$ was state s_0 reachable from the initial state. However, in $M^{\mathcal{D}}$, it is possible to switch realization after e.g. first transition to execute path $s \xrightarrow{r_1} s' \xrightarrow{r_3} s'' \xrightarrow{r_3} \ldots$ which allows reaching the state s_2. The over-approximation of the family given by the quotient MDP makes the state s_2 reachable, as outlined in Figure 3.3.

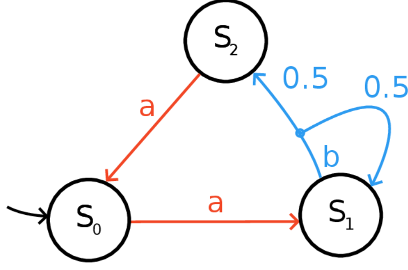


Figure 3.3: Illustration of the over-approximation of the quotient MDP – it is not possible to execute this path in neither MC of the family in Figure 3.2.

The scheduler which always selects realization $r \in \mathcal{R}^{\mathcal{D}}$ is called consistent. Checking the consistency of the scheduler decides if the MC induced by a given scheduler is a member of the family.

Definition 15 (Consistent scheduler). [3] Let $\mathcal{D} = (S, s_{\text{init}}, K, \mathcal{B})$ be a family of MCs and let $M^{\mathcal{D}} = \left(S, s_0, \mathcal{R}^{\mathcal{D}}, \mathcal{P}\right)$ be a quotient MDP of $\mathcal{D}$. For $r \in \mathcal{R}^{\mathcal{D}}$, a (memoryless) scheduler $\sigma_r \in \Sigma^{M^{\mathcal{D}}}$ is called r-consistent iff $\forall s \in S : \sigma(s) = r$. A scheduler is called consistent iff it is consistent for some $r \in \mathcal{R}^{\mathcal{D}}$.

The restriction of the quotient MDP wrt. subset of realizations $\mathcal{R} \subseteq \mathcal{R}^{\mathcal{D}}$ where $\mathcal{R} = \{r\}$ and $r \in \mathcal{R}^{\mathcal{D}}$ creates concrete MC of the family. Restricting the quotient MDP gradually for each $r \in \mathcal{R}^{\mathcal{D}}$ is another version of the one-by-one approach called the enumeration of consistent schedulers [10].

However, the abstraction approach is different. It is based on model-checking the quotient MDP. Because the model contains non-deterministic choices, the model-checking can provide only the lower v_{min} and upper v_{max} bound values and corresponding schedulers $\sigma_{\perp}$ and $\sigma_{\top}$. The interpretation of the model-checking results is crucial. Assume there is a target state $t \in S$ and single liveliness reachability property $\varphi_{>\lambda}$. If the maximum value v_{max} given by $\sigma_{\top}$ to reach the state t is lower than λ, then the entire family can be pruned, because no realization of the family can satisfy φ. If the v_{min} is greater than λ, then all realizations satisfy the property φ. When $v_{min} < \lambda < v_{max}$ and the scheduler $\sigma_{\top}$ is consistent, then the MC $M_{\sigma_{\top}}$ is the concrete member of the family. The MC $M_{\sigma_{\top}}$ satisfies property $\varphi_{>\lambda}$ as $v_{max} > \lambda$ and therefore the FSC $\mathrm{F}_{\sigma_{\perp}}$ is declared as the synthesis result. Nevertheless, in the case when $\sigma_{\top}$ is not consistent, the abstraction is too coarse and the refinement of the family is necessary. The reasoning about synthesis wrt. safety property is similar and requires checking the consistency of the $\sigma_{\perp}$. It can be also modified to max synthesis [10]. The upper and lower bound values are also returned to the learner because they can be used to prune larger design-space of FSC family $\mathcal{F}_{k+1}$.

Example 5. Assume the POMDP $\mathcal{M}$ 3.1, and safety property $\varphi_{<0.5}$ to reach the target state s_2. The model-checking of the quotient MDP $\mathcal{M}^{\mathcal{D}}$ provides the lower bound $v_{min} = 0$ and upper bound $v_{max} = 1$. It is true that $v_{min} < \lambda < v_{max}$, therefore, the consistency of $\sigma_{\perp}$ is checked. The $\sigma_{\perp}$ always select realization r_3 (see left-bottom of Figure 3.2) and therefore proves to be consistent. The FSC given by $\sigma_{\perp}$ is the result of the synthesis.

On the other hand, if the property would change to liveliness, e.g. $\varphi_{>0.5}$, then it turns out differently. The predicate $v_{min} < \lambda < v_{max}$ still holds and the scheduler $\sigma_{\top}$ is inconsistent. The Figure 3.3 shows $MC_{\sigma_{\top}}$ induced by inconsistent scheduler $\sigma_{\top}$. In this case, the abstraction is too coarse and thus it is needed to refine the family.

The refinement of the family is done by splitting it into two sub-families. Then. the quotient MDP of the sub-families can be created and the process follows recursively until the entire family is pruned or the feasible realization is successfully found.

Definition 16 (Splitting). [10] Let $\mathfrak{D}$ be a family of MCs, and $\mathcal{R} \subseteq \mathcal{R}^{\mathcal{P}}$ a set of realizations. For $k \in K$ and predicate A_k over S, splitting partitions $\mathcal{R}$ into

$$\mathcal{R}_{\top} = \{r \in \mathcal{R} \mid A_k(r(k))\} \quad \text{and} \quad \mathcal{R}_{\perp} = \{r \in \mathcal{R} \mid \neg A_k(r(k))\}.$$

Instead of splitting and rebuilding the quotient MDP in each iteration, we can make use of restricting the quotient MDP wrt. $\mathcal{R}_{\top}$ and $\mathcal{R}_{\perp}$. This is essential to the speed of the synthesis loop [10]. The synthesis speed can be accelerated by wisely choosing the predicate A_k according to which is the family split. The splitting strategy is based on the most significantly inconsistent parameter of the policy [6].

3.3 Counter-Example Guided Inductive Synthesis Method

The opposite approach to the AR method is the CEGIS method. While the AR method reasons about multiple realizations at once using the abstraction, the CEGIS is based on the analysis of a single realization. It takes one random realization and checks whether it satisfies a given property. If the property is satisfied, then the FSC which induces this realization is the result of the synthesis. Otherwise, a detailed analysis of the MC induced by this unsatisfiable realization is performed. The goal is, to find the critical part of the system, which causes the MC to be unsatisfiable. Essentially, the critical part is only a subset of states of the analyzed MC. All realizations, which behave the same in the critical part of the system can be safely pruned.

Definition 17 (Sub-MC). [3] Let $M = (S, s_0, P)$ be an MC with $s_{\perp} \notin S$ and let $C \subseteq S$ with $s_0 \in C$. The sub-MC of M wrt. C is an $MC\, M \downarrow C = (C \cup \{s_{\perp}\}, s_0, P')$, where the transition probability matrix P' is defined as follows:

$$P'\left(s'|s\right) = \begin{cases} P\left(s'|s\right) & \text{if } s, s' \in C, \\ 1 - \sum_{s'' \in S \setminus C} P\left(s''|s\right) & \text{if } s \in C \text{ and } s' = s_{\perp}, \\ 1 & \text{if } s = s' = s_{\perp}. \end{cases}$$

Definition 18 (Critical sub-system). [3] Let $M = (S, s_{\text{init}}, P)$ be an MC and let φ be a property s.t. $M \not\models \varphi$. If, for some set C, it holds $M \downarrow C \not\models \varphi$, then this set C and the corresponding subsystem $M \downarrow C$ are called critical. A critical set C is called minimal iff $|C| \leq |C'|$ for all critical sets C'.

The transition probability function in sub-MC re-routes all transitions leading out of a subset of states C to the absorbing state $s_{\perp}$. This changes the behavior of the states not included in the C and allows to study the behavior of the states in the subset C. If the MC $M \downarrow C$ induced by the subset C does not satisfy φ, then it is called a critical sub-system. In the synthesis of FSC for POMDPs, the induced MC is labeled as (s, n), indicating the state and the used memory. The set of relevant parameters is for each state (s, n) $\in$ C, the parameter (n, O(s)) $\in$ K. In other words, if the state (s, n) is in the critical-subsystem, then the parameter (n, O(s)) $\in$ K is called relevant [6]. The Algorithm 3 describes the fundamental approach of the CEGIS method.

Algorithm 3 Counterexample-guided inductive synthesis [3].

Input: A family of MCs, arbitrary property φ
Output: A realization $\mathcal{D}_r \models \varphi$ or `UNSAT`

1: $\mathcal{R} \leftarrow \mathcal{R}^{\mathcal{D}}$
2: **while** $\mathcal{R} \neq \emptyset$ **do**
3: $\quad r \leftarrow \text{any}(\mathcal{R})$
4: $\quad$ **if** $\mathcal{D}_r \models \varphi$ **then**
5: $\quad\quad$ **return** r
6: $\quad$ **end if**
7: $\quad C \leftarrow \text{criticalSubsystem}\,(\mathcal{D}_r, \varphi)$
8: $\quad \bar{K} \leftarrow \text{relevantParameters}(\mathcal{D}, C)$
9: $\quad \mathcal{R} \leftarrow \mathcal{R} \backslash (r \uparrow \bar{K})$
10: **end while**
11: **return** `UNSAT`

The minimal critical sub-system typically allows pruning a larger set of design space. However, the search for a minimal sub-system is not trivial and thus takes more time than the alternative approach. The alternative is, to search greedily from the initial state. The search begins with $C = \{s_0\}$ and the set of states C is then gradually enlarged until the property φ is not satisfied. To induce small counter-examples, the greedy approach takes into account parameters defining the set of realizations and prioritizes already relevant parameters [4]. The size of the set of relevant parameters is crucial. If the set of relevant parameters is large, then there is a smaller amount of realizations, which select these parameters. Therefore, the aim is to select as least parameters as possible. Another way to reduce the number of relevant parameters is to use the information about the above MDP acquired by the abstraction. Namely, the state-vector δ defining the upper or lower bounds to reach the target state from each state is computed. The upper (lower) bounds are utilized when the liveliness (safety) property is concerned. The vector δ is used to induce MC $M \downarrow C\,[\,\delta\,]$ – to reroute successor states of C to the state $s_\perp$. This state is considered the target state and the analysis of this induced MC takes that into account. Relevant parameters are still those belonging to set C, but the transitions of succ(C) are used in the greedy search, allowing the creation of smaller counter-examples.

Definition 19 (Counter-example). [6] A counter-example (CE) for FSC F and reachability property is a subset $C \subseteq S^F$ that induces the sub-MC $M \downarrow C\,[\,\delta\,]$ given as $(C \cup \text{succ}(C) \cup \{s_\perp, s_\top\}\,, (s_0, n_0)\,, P')$ where P':

$$P'(s) = \begin{cases} P^F(s) & \text{if } s \in C, \\ [s\top \mapsto \delta(s), s_\perp \mapsto 1 - \delta(s)] & \text{if } s \in \text{succ}(C) \backslash C, \\ [s \mapsto 1] & \text{if } s \in \{s_\top, s_\perp\}\,, \end{cases}$$

where $succ(C)$ is the set of direct successors of C, and the probability to reach $T \cup \{s_T\}$ from C is $< \lambda$.

This approach of finding critical sub-systems does not directly work for LRA properties. The reason for that and the adapted approach is explained in the next chapter.

3.4 PAYNT Tool

The PAYNT (Probabilistic progrAm sYNThesizer) was initially created for the automated synthesis of probabilistic programs. That is taking a partially implemented system with the holes and finding satisfying hole assignments. Nevertheless, is also supports the synthesis of FSC for POMDPs. As input, it takes POMDP specified in the *PRISM* or the *JANI* language, set of constraints and one optimization objective. The described methods (AR and CEGIS) are implemented in this tool. They were, however, implemented for the PCTL properties and the aim of this thesis is to extend *PAYNT* to synthesis wrt. LRA properties. The *PAYNT* is built on top of the *STORM* model checker and the Z3 solver is used to solve SMT formulas [5]. The AR utilizes the *STORMPY*[1] python API meaning that it is fully implemented in the PYTHON. The STORM supports model-checking of MDPs and MCs regarding LRA properties. Hence, the extension of the PAYNT to provide a possibility to perform FSC synthesis using AR method regarding LRA properties on POMDPs was technically already implemented. There was an issue with a method, which double-checked the results of the synthesis. It used wrong equation solver type, which did not work with LRA properties and thus was changed to default method. The one-by-one algorithm was implemented to provide a comparison to the AR method. Additionally, the possibility to generate statistics about synthesized families and quotient MDPs was added to PAYNT. Generated statistics were used to perform detailed analysis of the AR method described in Chapter 5. Nevertheless, most of the counter-example generation is written directly as storm extension. It is implemented in modern highly templated C++, which makes it challenging to add new functionality or make changes. Therefore, this work was not focused on the implementation of counter-example generation.

[1]STORMPY is available on github: https://github.com/randriu/stormpy/tree/synthesis

Chapter 4

Synthesis of Controllers Regarding Long-Run Average Properties

The problems encountered in many applications of POMDPs have a large timescale. The LRA properties are specifying the behavior of the agent in the environment regarding an infinite amount of time. Consequently, they are the subject of investigation. One use case is specifying the upper (lower) bound to the probability that the system is in a given state. For instance, ensure that the CPU will not be in a recovery mode more than 1% of the time. They are also utilized to specify that a given state is visited finitely or infinitely often because the LRA probability of a state that is visited finitely often is 0. E.g., make sure, that in mutual exclusion problem, each process is in the critical section infinitely many times. There are also expected long-run rewards (also known as mean-payoff) properties, where the reward function r: $S \times A \rightarrow \mathbb{R}$ assigns a real number reward to each state-action pair. However, they rely on techniques for computing the long-run probabilities [8, p. 830], thus, only those are considered.

The related work on finding a policy satisfying LRA specifications was made. In particular, the steady-state control (SSC) problem is defined as: Given an ergodic MDP and the goal steady-state distribution δ_{goal} over states, does there exist a policy which imposed on MDP yields MC whose steady-state distribution equals δ_{goal}? The policy is initially defined as history-dependent, but it is proven, that memoryless stochastic policies are sufficient to represent goal policy, provided it exists. The existence of the goal policy is decidable and the solution is provided by a linear program. These conclusions hold for the labeled MDP as well. The labeled MDP (LMDP) is considered because the concrete steady-state distribution is a very accurate specification. The states in LMDP are labeled to provide higher generalization over the state-space. The SSC problem is then finding the goal steady-state distribution over labels [2]. Given MDP is called ergodic, if every policy induces ergodic MC, and MC is ergodic if it is recurrent and aperiodic. The requirement of the ergodic MDP is later solved by defining another linear program, which induces a recurrent MC in possibly non-ergodic MDP. Additionally, there is a possibility to provide optimization objectives defined as expected rewards. This approach is called Steady-State Policy Synthesis (SSPS) [22].

For POMDPs, it was already mentioned, that the existence of policy specified with indefinite horizon property is undecidable [18]. The proof is based on extending known undecidable string-existence problems for probabilistic finite-state automata. The LRA properties belong to the indefinite horizon probabilities. Therefore, the SSC problem is

for POMDPs undecidable [2]. Nevertheless, the existence of an approximation problem is a recursively enumerable and therefore a decidable problem. The approximation problem for the optimization objective is: for a given POMDP, an objective function and $\varepsilon > 0$ – compute the optimal value within an additive error of ε. The approximation problem for threshold decision is defined in a similar fashion. The recursive enumeration is the consequence of the proof, that for every approximation problem, there is a finite-memory strategy that achieves the optimization objective within ε of the optimal value. The set of finite-memory strategies if finite, and therefore enumerable [11]. In this thesis, the set of finite-memory strategies represented by deterministic FSC will be searched for an ε-optimal solution, with the utilization of inductive synthesis methods introduced in the preceding chapter. In contrast with SSC, the specification is not the entire steady-state distribution over states (nor labels). To the best knowledge of the author, this is the first attempt of finding ε-optimal FSCs wrt. LRA objectives in POMDPs.

Problem statement. The problem is the same as in the previous chapter, but it assumes LRA properties: let the threshold $\lambda \in [0, 1]$, the set of operators $\bowtie \in \{<, >\}$, the set of target states $T \subseteq S$ and the set $\updownarrow \in \{min, max\}$. The problem is given by a POMDP $\mathcal{M}$, a set of constraints ϕ and an optimization objective $\boldsymbol{o}$. The set ϕ is such that $\forall \varphi \in \phi :$ $\varphi = LRA_{\bowtie\lambda}[T]$ and the optimization objective $\boldsymbol{o} = LRA_{\updownarrow}[T]$. The task is to construct the FSC F which imposed on POMDP yields MC satisfying the set of constraints ϕ and then incrementally improve the FSC F wrt. optimization objective $\boldsymbol{o}$.

The introduced problem can be solved with the use of the one-by-one Algorithm 2, but this chapter is focused on solving this problem with inductive controller synthesis methods, namely AR and CEGIS. They were introduced in the previous chapter and designed with the PCTL properties in mind. However, the key ideas can be extended to Long-Run Average (LRA) properties. The first section covers the integration of LRA properties into the abstraction-refinement method. The beginning of the second section explains why it is not possible to create a counter-example in the bottom strongly connected component. The rest of the section discusses techniques for generating counter-examples in the transient part of the MC states.

4.1 Integrating LRA Properties to the AR Method

The core of the abstraction-refinement method is to verify the behavior of an entire family of controllers represented by a quotient MDP. If the verification is inconclusive, then the family is split to create a less general family. This concept was discussed in detail in the preceding chapter. Integration of Long-Run Average (LRA) properties to abstraction-refinement method consists of allowing model-checking abstraction MDP with respect to LRA properties. The interpretation of the model-checking result was explained in the previous chapter wrt. reachability property and it is exactly the same for the LRA properties. Therefore, it will not be explained again – instead the Algorithm 4 defining this approach is provided.

The following example demonstrates Algorithm 4.

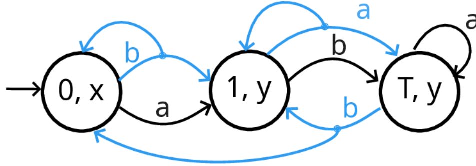
Example 6. Assume the POMDP $\mathcal{M}$ outlined in Figure 4.1 and the LRA property $\varphi =$ LRA$<$ 0.27 [T]. The 1-FSC family $\mathcal{F}_1$ consists of 4 controllers given by combinations of one of two actions (a, b) for both observations (x, y). Under the POMDP $\mathcal{M}$, there is the quotient MDP $M^{\mathcal{D}}$ given by 4 realizations $\mathcal{R} = \{r_1, r_2, r_3, r_4\}$ defined by corresponding controllers. E.g., the r_2 corresponds to the controller which selects an action in observation

Algorithm 4 Abstraction-Refinement Algorithm (Adapted [3, Alg. 4] wrt. LRA props.)

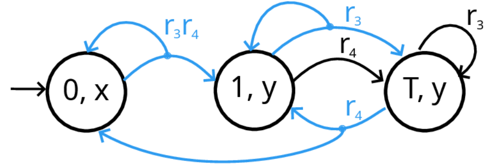
Input: POMDP $\mathcal{M}$, family of k-FSC $\mathcal{F}_\mu$, a LRA safety property $\varphi = LRA_{<\lambda}[T]$
Output: A satisfying FSC or `NONE`

1: $M^{\mathcal{D}} \leftarrow$ buildQuotientMDP$(\mathcal{M}, \mathcal{F}_\mu)$ ▷ Using Definition 14
2: $\mathfrak{F} \leftarrow \{\mathcal{F}_\mu\}$
3: **while** $\mathfrak{F} \neq \emptyset$ **do**
4: $\quad \mathcal{F} \leftarrow$ any$(\mathfrak{F})$
5: $\quad \mathfrak{F} \leftarrow \mathfrak{F} \setminus \{\mathcal{F}\}$
6: $\quad \mathcal{R} \leftarrow$ realizations$(\mathcal{F}, \mathcal{M})$
7: $\quad M \leftarrow M^{\mathcal{D}}[\mathcal{R}]$ ▷ Restrict MDP using Def. 14
8: $\quad (\boldsymbol{v}_{\min}, \sigma_{\min}, \boldsymbol{v}_{\max}, \sigma_{\max}) \leftarrow$ boundsLraMdp(M, T) ▷ Using Definition 7
9: $\quad$ **if** $\boldsymbol{v}_{\min} > \lambda$ **then**
10: $\quad\quad$ **continue** ▷ Reject all family members
11: $\quad$ **end if**
12: $\quad$ **if** $\boldsymbol{v}_{\max} \leq \lambda$ **then**
13: $\quad\quad$ **return** $any(\mathcal{F})$ ▷ $\forall r \in \mathcal{R} \vDash \varphi$
14: $\quad$ **end if**
15: $\quad$ **if** $\sigma_{\min}$ is r-consistent for some $r \in \mathcal{R}$ **then** ▷ Using Definition 15
16: $\quad\quad$ **return** FSC $F_{\sigma_{\min}}$
17: $\quad$ **end if**
18: $\quad (\mathcal{F}_\top, \mathcal{F}_\bot) \leftarrow$ split$(\mathcal{F})$ ▷ Split family of FSCs
19: $\quad \mathfrak{F} \leftarrow \mathfrak{F} \cup \{\mathcal{F}_\bot, \mathcal{F}_\top\}$
20: **end while**
21: **return** `NONE`

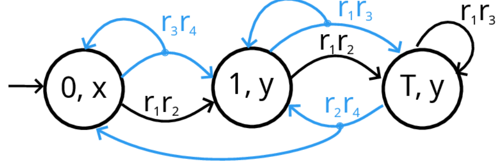
POMDP$\mathcal{M}$



MDP$M^D|R_\top|$



MDPM^D



MDP$M^D|R_\bot|$

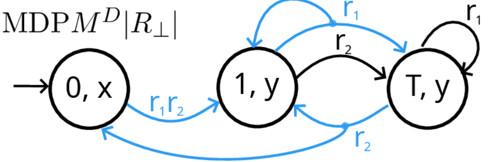


Figure 4.1: Top-left: POMDP $\mathcal{M}$ with two observations and two actions. States are labeled as (n, z) where $s_n \in S \wedge O(s_n) = z$. All actions have either probability 1 or 0.5 and 0.5, therefore are omitted from drawings. Bottom-left: Quotient MDP $M^{\mathcal{D}}$ induced by 4 realizations given by 1-FSC family. Right: Quotient MDP $M^{\mathcal{D}}|\mathcal{R}_\top|$ and $M^{\mathcal{D}}|\mathcal{R}_\bot|$ induced by restricting $M^{\mathcal{D}}$ wrt. predicate selecting an action in s_0. Blue actions are avoiding the target state T the most.

z and action b in observation y. The analysis result of this quotient MDP provides lower and upper bounds $v_{min} = 0.25$ and $v_{max} = 1$ (7th line Alg. 4). The analysis is inconclusive because $v_{min} < 0.27 < v_{max}$ and therefore, the consistency of scheduler σ_{min} is checked. If the scheduler were consistent, then the MC $M_{\sigma_{min}}$ would be MC induced by some realization r and would satisfy property φ. However, that is not the case – in the $M^{\mathcal{D}}$, the blue actions that are least likely to lead to the state T are selected by σ_{min}. It is evident, that neither r_n selects all these actions, therefore the σ_{min} is inconsistent and the family $\mathcal{F}_1$ is split into two sub-families (17th line Alg. 4) by parameter selecting an action in observation x. The set of realizations is split to $\mathcal{R}_\top$ and $\mathcal{R}_\bot$ and 2 restricted quotient MDPs $M^{\mathcal{D}}|\mathcal{R}_\top|$ and $M^{\mathcal{D}}|\mathcal{R}_\bot|$ arises. Then, the process is repeated – analysis of $\mathcal{R}_\top$ is again inconclusive, but analysis of $\mathcal{R}_\bot$ has lower bound $v_{min} = 0.29$ and thus, the entire family can be pruned (9th line Alg. 4). Note that the analysis using the AR method highly depends on the λ value. If the λ would equal 0.2, then 1 iteration would be enough to prune all realizations. On the other hand, if the $\lambda = 0.35$, then the analysis of $\mathcal{R}_\bot$ would be inconclusive too, increasing the number of required iterations. In general, if the λ is too high or too low, then is the AR method superb.

When the entire family of $\mathcal{F}_k$ controllers is pruned and the satisfying controller is not found (or the optimization objective is specified), then the family $\mathcal{F}_{k+1}$ is created and the Abstraction-Refinement method is used on the larger family, see Algorithm 1. In the case of optimization objective, the best-found value is passed to AR with the $\mathcal{F}_{k+1}$. The value can be used to prune the sub-family, which optimal value does not reach the optimum value from the previous family. The following example illustrates the effect of adding memory nodes on the optimality value.

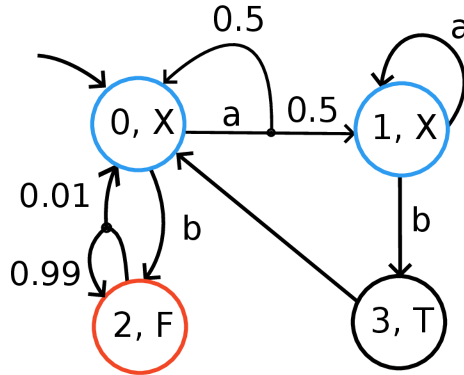


Figure 4.2: MemoryDemo: The POMDP $\mathcal{M}$ with states labeled as (n, z) where $s_n \in S \wedge O(s_n) = z$. There is only 1 choice between a and b in observation x (blue). It is designed to show that more memory can improve the optimization LRA property.

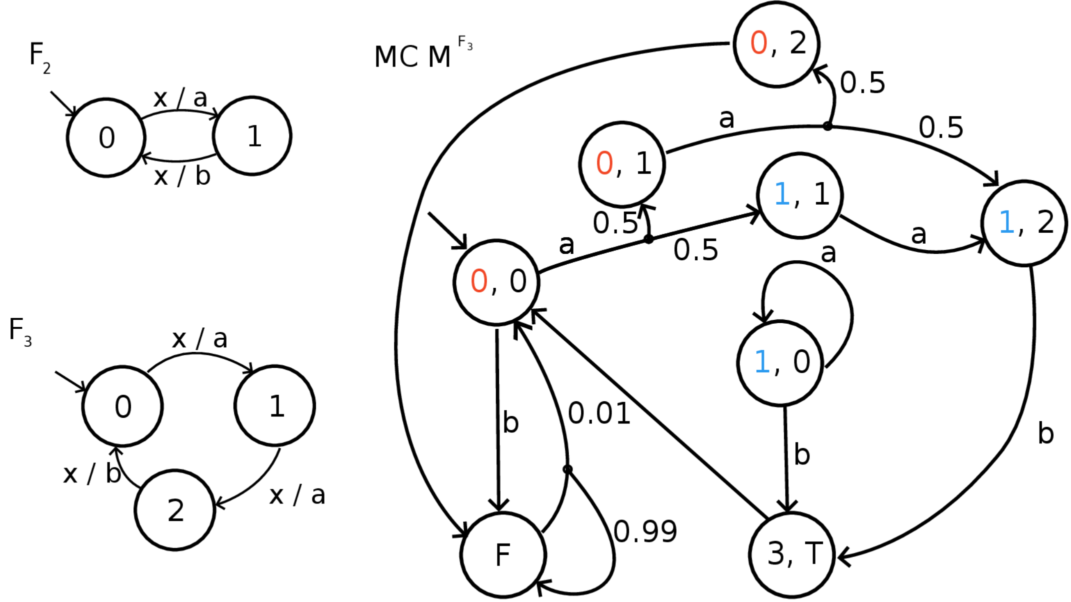


Figure 4.3: Top-left: Part of optimal FSC F_1. Bottom-left: Part of optimal FSC F_2. Right: MC induced by optimal FSC F_1. States are labeled as (x, n) where s_x is the state and n is the memory node. The unreachable state is colored gray.

Example 7. Assume the POMDP $\mathcal{M}$ from MemoryDemo problem outlined in Figure 4.2 and the goal to be in the state T as much as possible – optimality property LRA_{max} [T]. State 2 has observation F and once reached, the probability to leave this state is small, this models punishment to playing action b in state 0. In contrast, action b in state 1 leads to the target state T. Critically, states 0 and 1 have the same observation x. The memoryless controller F_1 decides if the action a or b should be played in observation x. Notice that reaching state T requires playing different actions in the same observation, therefore, the state T is unreachable with the memoryless controller. States 2 and 3 are perfectly observable because they have unique observations. The memory is ineffective in those states, hence the memory function $\mu(F) = \mu(T) = 1$ and $\mu(x) = 2$ is used. The part of best FSC F_2 and F_3 from family $\mathcal{F}_2$ and $\mathcal{F}_3$, respectively, are in the Figure 4.3. The observations F and T have only one possible action and therefore are omitted in both controllers. Notice that, using F_2, the state T is now reachable because the memory node is allowed to play action a and then action b in the observation x. However, there is still a significant chance to stay in state s_0 (by playing a) and then transition to punishment state F (by playing b). Imposing FSC F_3 on POMDP $\mathcal{M}$ creates MC M^{F_3} (see Figure 4.3). The third memory node allows playing action 2 times, to increase the chance to be in state s_1 (blue) when playing action b to hopefully transition to state T. Adding another more memory nodes allows to improve the controller even more in the same manner. Eventually, it would not be profitable to play another action a, because the chance to be in state s_1 is sufficiently high and the optimization objective is to maximize being in state T. In the next chapter, the result of the experimental evaluation is that more than 7 memory nodes do not improve the optimum.

Model-checking a quotient MDP of the given family often takes more time than it takes to model-check the concrete MC of that family because the MDP contains non-determinism. Additionally, concrete MC is smaller as it does not contain unreachable states. For the AR

to be effective, it must hold, that sum of time periods to model-check the abstraction MDP and refined sub-MDPs is smaller than model-checking each MC separately. This constraint is always satisfied when considering reachability properties. However, when it comes to LRA properties, that is not always the case. Algorithms to obtain LRA properties are based on computing a large number of time steps till the steady-state of the system is reached, i.e. the algorithm converged. Small transition probabilities in a model can also negatively impact the time for the algorithm to converge because the number of required iterations increases. The above reasons may reduce the speed of AR so much that one-by-one enumeration is faster, as is demonstrated in the following chapter.

4.2 Generating LRA Property Counter-Example

The counter-examples for FSCs and the CEGIS algorithm are well defined in the previous chapter. The crucial component of the CEGIS algorithm is finding a critical sub-system, which suffices to refute the specified property. The critical-sub system is thereafter used to select relevant parameters and prune the subset of realizations. This section focuses on finding critical-sub systems regarding LRA properties. The LRA and reachability properties are fundamentally different in the changes of behavior of the derived sub-systems. The generation of counter-example regarding reachability property is based on finding a path, which refutes the property. The set of states on the paths forms a critical sub-system, where all transitions that don't belong to the path are rerouted to the $s_\perp$ state. In this sub-system, the reachability property does not change. However, rerouting any transitions has a significant impact on the LRA properties. The transient states of any MC are visited only finitely many times, therefore, the LRA to be in a transient state is 0. The non-transient state is called recurrent and states reachable from a recurrent state form a recurrent class. The steady-state distribution of the recurrent aperiodic class is obtained by solving the set of linear equations, where every transition depends on the final steady-state distribution. Consequentially, the rerouting in the recurrent class has an impact on the LRA behavior of the sub-system. In the ergodic Markov chain, all states form a single recurrent class and therefore it is not possible to create counter-examples in ergodic MCs.

In non-ergodic MCs, obtaining the steady-state distribution of the entire MC is based on computing the steady-state distribution for each recurrent class and multiplying them by the probability to reach the recurrent class. Because of that, the generation of critical sub-systems wrt. LRA properties are based on reachability properties to individual recurrent classes. In graph theory, a recurrent class is called the bottom strongly connected component (BSCC). A given MC is partitioned to the set of BSCCs and the LRA property is calculated separately for each one, by declaring any state belonging to the BSCC as initial. Let's assume a MC M, a single target state $t \in S$, and a safety LRA property $\varphi = LRA_{<0.2}[t]$. The MC contains several BSCCs and the target state t belongs to BSCC $\mathcal{B}_t$. Assuming only the states in the target BSCC, the LRA is, let's say, 0.5. If the probability to reach $\mathcal{B}_t$ was 1 – there was no other BSCC except $\mathcal{B}_t$ – then the final LRA would be 0.5. If the probability to eventually reaching $\mathcal{B}_t$ was 0.8, then the final LRA to be in state t would be $0.8 * 0.5 = 0.4$. Whenever is the probability $\boldsymbol{p}_t$ of reaching the $\mathcal{B}_t$ high enough that when multiplied by the LRA in $\mathcal{B}_t$ exceeds the value of φ, then it is sufficient to refute the property φ. I.e., if $\boldsymbol{p}_t \cdot 0.5 > 0.2$ is true, then in refutes the φ. In this moment, the generation of critical subsystem wrt. LRA property can be transformed to the generation wrt. reachability property $\varphi_2 = P_{<\lambda}[T]$ where $T \subset S$ is a set of states which belong to the BSCC $\mathcal{B}_t$ and the λ value, is given by dividing the value of LRA property φ by the

calculated LRA in the $\mathcal{B}_t$. In the discussed example, the $\lambda = 0.2/0.5 = 0.4$ and note that if $\boldsymbol{p}_t$ is greater then λ then the LRA property φ is refuted.

However, the transformation to the reachability property works only in the case, where all target states belong to the single BSCC. Therefore, a more generalized approach is assumed, which allows consideration of multiple target states across different BSCCs. It is accomplished by *collapsing* the states belonging to any BSCC to the absorbing state, which represents the corresponding BSCC. In this collapsede MC, the critical sub-system is found with the knowledge of LRA properties in each individual BSCC. During the generation, the probability to reach each BSCC $\mathcal{B}_i$ is multiplied by the LRA value in $\mathcal{B}_i$. Adding up these values can refute the given LRA property. The algorithm of this approach and the formal model for collapsed MC is described below. In addition, this approach can be extended by incorporating the abstraction element. The MC is still induced by the underlying quotient MDP. By folding the states belonging to any BSCC in the MDP, the collapsed MDP arises and the upper (lower) bounds can be obtained by maximizing (minimizing) policy. Then, those bounds are used to create rerouting with state $s_\top$ and essentially generate smaller counter-examples.

4.2.1 Critical sub-systems in Ergodic MCs

Creating a critical sub-system to a reachability property and a given MC consists of finding a path, which suffices to refute an investigated property. The probability to take the path $s_0 \to \cdots \to s_n$ is calculated by multiplying the transition probabilities between the states on the path, i.e. $\Pi_{x=1}^{n} P(s_x|s_{x-1})$. Note, that calculating the probability to take a given path does not include any other paths in the MC. Adding the paths starting in the initial state s_0 and ending in the target state s_n can only increase the probability of eventually reaching the target state. In other words, adding any other path does never decrease the reachability property. This allows to take states on the path and declare them as critical sub-system. Unfortunately, this is not the case when the LRA probabilities are concerned.

Proposition 1. It is not possible to construct a critical sub-system for a given ergodic MC and an LRA property.

Let's assume an MC M where each state is reachable from any other state – states form a single recurrent class. The LRA to be in a given subset of target states is defined using the steady-state distribution. Recall (from Definition 4) that the distribution is given by the set of linear equations, where each path of the MC is concerned. That means, that each path of the MC does have an impact on the final distribution. Additionally, in this MC, changing the initial state does not change the distribution. This contrast between the reachability and LRA properties has a direct consequence – there isn't a path between any states which can refute the LRA property. Generation of counter-examples for reachability properties was explained in the previous chapter using the rerouting – transitions which are not concerned are rerouted to the absorbing state $s_\perp$. Such rerouting of any transition decreases or increases the LRA probability, and therefore using rerouting in recurrent class is not an option.

4.2.2 Generating counter-examples in transient states of MCs

While the generation of a critical sub-system is not possible in ergodic MCs, it is possible in Markov chains containing transient states. The first step is computing LRA property in

each bottom strongly connected component (BSCC) seperately. That is accomplished by decomposing a MC to the set of bottom strongly connected components.

Definition 20 (BSCC). [15, 8] Let M be a finite MC, the strongly-connected component (SCC) is a subset of states $C \subseteq S$ where for each $s, s' \in C$ exists a path from s to s'. A bottom SCC (BSCC) of M is an SCC B from which no state outside B is reachable, i.e. $\forall s \in C$ and $s' \in S \setminus C$: $P(s, s') = 0$. Let $BSCC(M)$ denote the set of all BSCCs of the MC M.

If the initial state of a given MC is changed to any state belonging to some BSCC, then states belonging to this BSCC are only reachable states. This change creates ergodic MC and it was already mentioned, that the steady-state distribution of ergodic MC is independent of the initial state. The steady-state distribution for each ergodic MC created by changing the initial state to any state of BSCCs is obtained. Then, the LRA property is calculated (see Definition 5) by adding up the steady-state values of target states. The LRA value wrt. target state is computed for each BSCC, obtaining the BSCC-vector $\boldsymbol{lraBSCC}$ of LRA values in each BSCC.

The LRA behavior in each BSCC is computed, therefore, the initial MC is reduced wrt. the set of BSCCs. The reduction is based on *collapsing* the states, which belong to any BSCC to one state, which represents the corresponding BSCC. In other words, for each BSCC $\mathcal{B}_i \in BSCC(M)$ is created one state $\overline{s}_i$. Then, each transition that leads to state $s \in \mathcal{B}_i$ is rerouted to state $\overline{s}_i$. Additionally, all states $\overline{s}_i$ are absorbing, which simulates the behavior of BSCC. Set S^{inbscc} created using predicate inbscc(s) is utilized to create collapsed MC.

Definition 21. Let $M = (S, s_0, P)$ be a MC and BSCC(M) be a set of BSCCs of M. The predicate inbscc is true if a given state s belongs to any BSCC, i.e., inbscc(s) iif $\exists\, B_i \in BSCC(M) : s \in B_i$. The set $S^{inbscc} \subseteq S$ contains all states satysying the predicate inbscc, that is $S^{inbscc} = \{s \in S \mid inbscc(s)\}$.

Definition 22 (Collapesd MC). Let $M = (S, s_0, P)$ be a MC, BSCC(M) a set of BSCCs of M and $k = |BSCC(M)|$. The collapsed MC $M_{\mathcal{B}} = (S_{\mathcal{B}}, s_0, P_{\mathcal{B}})$ is reduced wrt. BSCC(M). The $S_{\mathcal{B}} = S \setminus S^{inbscc} \cup \{\overline{s}_1, \overline{s}_2, \ldots, \overline{s}_k\}$, where each $\overline{s}_i$ represents its corresponding BSCC $\mathcal{B}_i \in BSCC(M)$. The $P_{\mathcal{B}}$ is defined for each pair $s, s' \in S_{\mathcal{B}}$ as:

$$P_{\mathcal{B}}(s'|s) = \begin{cases} 1 & \text{if } s = s' = \overline{s}_i, \\ \sum_{s'' \in B_i} P(s''|s) & \text{if } s \neq s' \wedge s' = \overline{s}_i, \\ P(s'|s) & \text{otherwise.} \end{cases}$$

At this point, the bscc-vector describing the LRA behavior of each BSCC component and collapsed MC $M_{\mathcal{B}}$ is created. Searching for subset $C \subseteq S_{\mathcal{B}}$ of critical transient states that form a critical sub-system in $M_{\mathcal{B}}$ is following. Let the $C = \{s_0\}$ contain only the initial state and create the sub-MC $M_{\mathcal{B}} \downarrow C$ using the Definition 17. All transitions which do not end in any state $s \in C$ are rerouted to the absorbing bottom state $s_{\perp}$. Then, the probability of eventually reaching each $\overline{s}_i$ from initial state s_0 is calculated. As each $\overline{s}_i$ represents B_i, another bscc-vector $\boldsymbol{reachBSCC}$, indicating the probability to eventually reach B_i, is obtained. The LRA behavior of sub-MC $M_{\mathcal{B}} \downarrow C$ is obtained by multiplying each LRA $\mathcal{B}_i$ with each corresponding $\overline{s}_i$ and adding that up. This is known as the dot product of vectors. If the obtained value exceeds the λ value of a safety LRA property $\varphi = LRA_{<\lambda}$, then the subset of states C is enough to refute the property φ. However,

the C is a subset of states in collapsed MC $M_{\mathcal{B}}$, but the subset of states original MC M is needed. Therefore, all reachable states $\overline{s}_i \in C$ must be replaced with states $s \in \mathcal{B}_i$ that were collapsed in the original MC. In the case, when the subset C does not refute the property φ, then it is enlarged by including any state that is reachable from C. The described approach is outlined in the Algorithm 5.

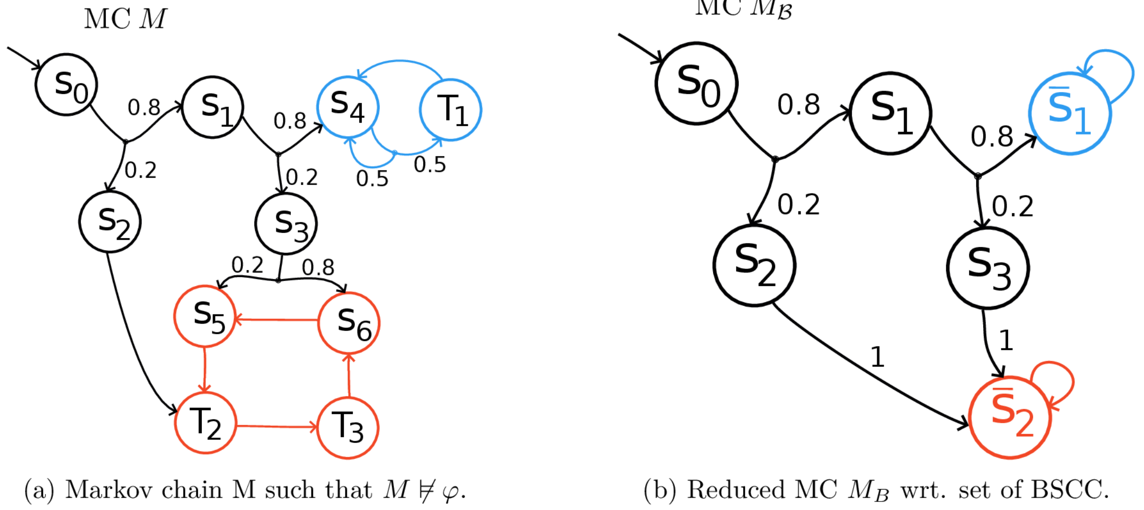
Algorithm 5 Generation of a critical subsystem wrt. a LRA property φ

Input: A MCs M s.t. $M \not\models \varphi$, a LRA safety property $\varphi = LRA_{<\lambda}[T]$
Output: A critical set $C \subseteq S$ for M and φ

```
1:  𝓑 ← BSCC(M)                                        ▷ Using Definition 20
2:  lraBSCC ← [∀𝓑_i ∈ 𝓑 : LRA(𝓑_i, T)]
3:  M_𝓑 ← collapseMC(M, 𝓑)                             ▷ Using Definition 22
4:  C_0 ← {s_0}, x ← 0
5:  S ← S_𝓑 \ C_0
6:  while S ≠ ∅ do
7:      M_C ← M_𝓑 ↓ C_x                                 ▷ Using Definition 17
8:      reachBSCC ← [∀𝓑_i ∈ 𝓑 : reachabilityMC(M_C, s̄_i)]
9:      if lraBSCC · reachBSCC > λ then
10:         C ← C_x \ {s̄_i ∈ S_𝓑}
11:         return C ∪ {s ∈ 𝓑_i | reachBSCC(𝓑_i) > 0}
12:     end if
13:     s ← reachableState(C_x)
14:     S ← S \ {s}
15:     C_{x+1} ← C_x ∪ {s}               ▷ Add random reachable state to critical states
16:     x = x + 1
17: end while
```

The worst case scenario is gradually including all states $s \in S_{\mathcal{B}}$ to subset C. When $C = S_{\mathcal{B}}$, then no transition is rerouted in the sub-MC $M_{\mathcal{B}} \downarrow C$ and therefore it has the same behavior as MC M. Consequentially, the proposed algorithm will always terminate. Let's assume a MC M_r induced by a random realization (see Algorithm 3, 4th line). The M_r is depicted in Figure 4.4a and it will be referred to as MC M. Assume an LRA safety property $\varphi = LRA_{<0.1}[T]$. The analysis of this MC provides a result $LRA[T] = 0.39$, which is greater than 0.1 and therefore $M \not\models \varphi$. Therefore, the Algorithm 5 is initiated to find a critical subsystem for M and LRA property φ. The following example will demonstrate the application of the algorithm.

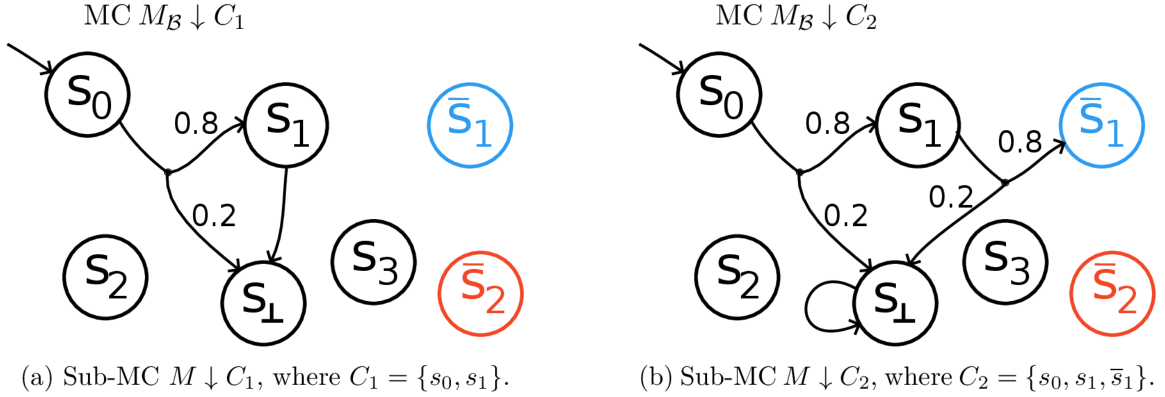
Example 8. The decomposition of BSCC is performed and two BSCC $\mathcal{B}_1, \mathcal{B}_2 \in BSCC(M)$ are created. In the Figure 4.4a, $\mathcal{B}_1$ is colored blue and $\mathcal{B}_2$ red. The initial state is changed to $s_4 \in \mathcal{B}_1$ and then, the LRA [T] is found to be 1/3. This is repeated for each $\mathcal{B}_i$ (see Algorithm 5, 2nd line) and thus, the vector $\boldsymbol{lraBSCC} = [1/3 \;\; 1/2]$ is created. Next, the collapsed MC $M_{\mathcal{B}}$ is constructed using Definition 22 (and is depicted in Figure 4.4b). The corresponding states $\overline{s}_1$ and $\overline{s}_2$ representing BSCCs $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively, are created. The sets $C_0 = \{s_0\}$ and $C_1 = \{s_0, s_1\}$ do not create sub-MCs, which refutes the given property. The collapsed sub-MC $M_{\mathcal{B}} \downarrow C_1$ using the set $C_1 = \{s_0, s_1\}$ is depicted in the Figure 4.5a. For the $M_{\mathcal{B}} \downarrow C_1$, the vector $\boldsymbol{reachBSCC} = [0, 0]$, because neither state $\overline{s}_i$ is reachable. Therefore, the set C is enlarged by selecting a random reachable state $\overline{s}_1$ meaning that $C_2 = \{s_0, s_1, \overline{s}_1\}$. The reduced MC $M_{\mathcal{B}} \downarrow C_2$ is depicted in Figure 4.5b and the vector $\boldsymbol{reachBSCC} = [0.64 \;\; 0]$. The dot product of vectors $\boldsymbol{lraBSCC}$ and



(a) Markov chain M such that $M \not\models \varphi$.

(b) Reduced MC $M_{\mathcal{B}}$ wrt. set of BSCC.

Figure 4.4: The MC M, where color distinguishes which state belongs to which BSCC and the reduced MC $M_{\mathcal{B}}$ with created $\overline{s}_i$ states, representing the corresponding BSCC.



(a) Sub-MC $M \downarrow C_1$, where $C_1 = \{s_0, s_1\}$.

(b) Sub-MC $M \downarrow C_2$, where $C_2 = \{s_0, s_1, \overline{s}_1\}$.

Figure 4.5: The set of critical C_n set is gradually increased till it suffices to reject φ.

reachBSCC is $0.21\overline{3}$ which exceeds the φ value of 0.1 and therefore is sufficient to refute φ. The result of the algorithm is set C_2, where each $\overline{s}_i$ is replaced by states which belong to the corresponding BSCC $\mathcal{B}_i$.

The Algorithm 3 then continues on the 8th line to select relevant parameters based on the obtained subset of critical states. Then, the set of all realizations is pruned wrt. relevant parameters. The previous chapter explained, why smaller critical subsystems induce a smaller number of relevant parameters and therefore enable pruning of the larger subset of realizations. The abstraction approach was used to create smaller counter-examples wrt. reachability properties and the following subsections show how to utilize abstraction to create smaller CEs wrt. LRA properties.

4.2.3 Towards smaller counter-examples using abstraction

The counter-example generation is based on taking random unsatisfiable realization and generalizing this realization to a larger subset of realizations, which can be safely pruned from the design-space. In the process of finding a critical sub-system, the simplest approach

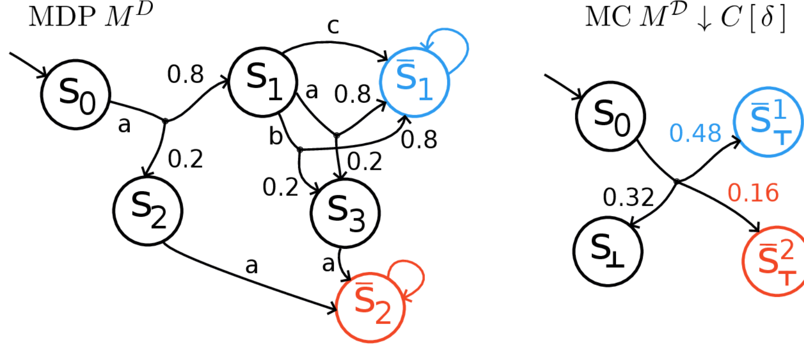


Figure 4.6: The reduced quotient MDP wrt. BSCC(M) where M is MC from Figure 4.4a.

forgets that the realization is part of some larger system and instead considers only the MC induced by one realization. However, the information about the above quotient MDP can be certainly used to generate smaller critical sub-systems. The analysis of the MDP provides lower and upper bounds. The bounds are used to re-route all transitions leading out of critical sub-system C to freshly created state $s_\top$, which is added to the subset of target states. This concept can be extended to LRA properties.

The underlying unsatisfiable MC M is decomposed to the set of BSCCs and the LRA in each BSCC is computed. Then, instead of *folding* states in the MC to create reduced MC $M_\mathcal{B}$, the states in the quotient MDP are folded to create the *reduced MDP*. The reduction takes into account the decomposition BSCC(M) of a given unsatisfiable MC M. Similarly, all actions leading to any state $\mathcal{B}_i \in BSCC(M)$ are rerouted to state $\overline{s}_i$ which represents the corresponding BSCC. Then, the minimum or maximum bounds (depending on safety or reachability property) to reach the folded states $\overline{s}_i$ are calculated for each state to obtain state-vector δ. This vector is thereafter used to induce sub-system $M \downarrow C[\,\delta\,]$, which reroutes transitions leading out of C to state $s_\top$, which is considered the target state. However, the state $s_\top$ is necessary for each $\overline{s}_i$, because the reachability bounds to each BSCC must be calculated. The following example demonstrates this idea.

Example 9. Recall the MC M induced by the random realization of quotient MDP $M^\mathcal{D}$, the decomposition BSCC(M) and reduction of MC M is in Figure 4.4. The the vector $\boldsymbol{lraBSCC} = [1/3 \;\; 1/2]$ is same as in the previous example. Similarly, the reduced MDP $M^\mathcal{D}$ wrt. the same set BSCC(M) is in Figure 4.6. The same LRA property $\varphi = LRA_{<0.1}[\,T\,]$ is assumed, it is the safety property, therefore the state-vector δ contains minimum bounds. Take a look at the state initial state s_0, the minimum probability of eventually reaching $\overline{s}_1$ and $\overline{s}_2$ is 0.45 and 0.2, respectively. Using this values and $C = \{s_0\}$, the MC $M^\mathcal{D} \downarrow C[\,\delta\,]$ arises. The minimum probabilities are rerouted to corresponding states $s_\top^i$. The reachability probabilities bscc-vector, to reach target states is $[0.48 \quad 0.16]$ and the dot product is 0.24 and which is already larger than φ value 0.1. Therefore, critical sub-system C containing only one state is enough to refute φ. Notice that the sub-system C is much smaller than the sub-system from the previous example, where abstraction was not used. Namely, state s_1 is not included in the sub-system, and therefore, all realizations, which select action a in state s_0 can be pruned. Instead of punning just realizations selecting action a in s_0 and b in s_1, as was the case in the previous example.

Chapter 5

Experimental Evaluation

The goal of this chapter is to answer the following research questions:

Q1: **Are synthesis methods effective when considering LRA properties?** Controller synthesis methods were initially created with PCTL properties in mind. Key ideas behind inductive controller synthesis methods can be utilized to synthesis with respect to long-run average properties, as described in the previous chapter. The effectiveness and applicability of the AR method are studied throughout this chapter.

Q2: **How much memory is needed to solve simple POMDP problems?** Every POMDP has finite memory ϵ-optimal strategy for LRA objectives [11]. Searching for optimal strategy starts at FSC with one memory node, then the memory is incrementally added until the optimal strategy is found. Adding many memory nodes yields harder synthesis problems as a design-space increases significantly. However, solving simple POMDP problems requires only a few memory nodes (see Section 5.2).

Q3: **How much is the AR method faster in comparison with the one-by-one considering LRA properties?** A benchmark containing several experiments was created and synthesis was run using both methods to evaluate their efficiency. The one-by-one controller synthesis method is used as a baseline algorithm and it is confirmed that it does not scale on large problems. Both methods do always provide the correct (optimal) solution, therefore execution time is the sole relevant factor.

Q4: **How does the speed of synthesis compare between the reachability and LRA properties on a given model?** Compared to the baseline algorithm, the AR method is always faster in synthesis with specifications containing PCTL properties. The comparison in acceleration (see Section 5.3) on a given model with different types of specifications allows to show that the performance is comparable when LRA properties are concerned.

Q5: **Why does the one-by-one method sometimes outperform the AR and can we prevent it?** While the AR is faster in most cases, there is one special model, where the one-by-one method is better on LRA specifications. In Section 5.4, the impact of design-space, state-space, and transition probabilities on the AR speed is investigated, and a few improvement ideas are proposed.

problem name	|S|	|A|	|Z|
storm-problem	3	5	2
memory-demo	4	6	3
endless-maze	18	138	7
endless-maze-large	27	110	7
blind nanny	27	90	15
robot-battery	65	122	13
robot-battery-stay	71	195	13
drone-4-1	1125	2954	383
crypt4	2068	4708	558

Figure 5.1: Comparison of the sizes of the benchmark experiments.

5.1 Benchmark details and models introduction

The benchmark was run on AMD Ryzen 7 5700G (3800 MHz, 16 cores) with up to 32GB of RAM and all used algorithms are single-threaded. The number of states, actions, and observations of all benchmark problems are summarized in Figure 5.1. Experiments[1] were designed with a particular intent of not having an absorbing goal state because otherwise, the LRA properties of the model are trivial. Instead, an objective of experiments is to reach a specific subset of target states infinitely often. One way to acquire that is to provide a reason to leave a goal state, e.g. robot must leave a subset of goal states to go charge itself, etc. Another is providing liveliness specification of two different goal states, with non-zero path length between them. There are such experiments in the created benchmark:

- *Robot-battery* is the first problem in a benchmark – the robot is attempting to maximize time spent exploring the state while also keeping its battery away from running out. Robot has to go charge itself once in a while so the battery will not discharge. However, the robot only sees if it is in a charging state, an exploring state, or neither. It is also provided with a power manager which tells the robot what the battery level is – full battery, high, low, or discharged. There are 9 observations in total and 2 actions (left and right) therefore the design-space describes $2^9 = 512$ options.

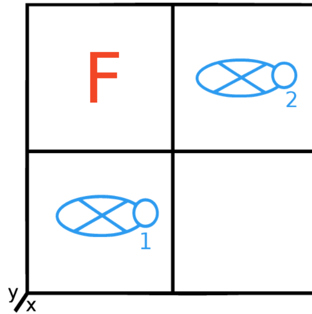
- *Robot-battery-stay* is an extended version of the Robot-battery problem. The action *stay* is introduced to provide the robot the possibility to remain in a current state in the grid (while the battery is running out). Therefore, the design-space is deliberately increased to $3^9 = 19683$ different FSCs.

- The second model is *memory-demo* outlined in Figure 4.2. The problem is designed to show on a small model that adding memory to FSC improves the optimum.

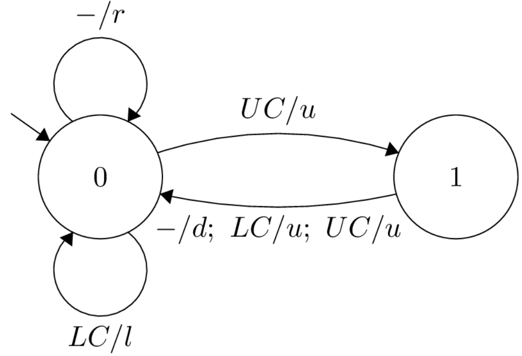
- *storm-problem* is another very small POMDP. Therefore, the synthesis problem is included twice in the benchmark – with memory 4 and 5. When the memory is added, the design-space explodes to 16 384 and 500 000 options in a family of 4-FSC and 5-FSC, respectively.

- The *blind-nanny* problem is described in Figure 5.10. As the number of total observations is 12 and there are 4 actions at each state (l, r, d, u), there are 4^{12} =16 777

[1] Experiments are published on https://github.com/AntoninJarolim/synthesis/tree/4db61e69bfce65e6cdadf4c661d7e1ddc86c0e1c/models/pomdp/no-goal-state.



(a) Small version of *blind-nanny* problem.



(b) Synthesized 2-FSC: UC, LC, and '-' means upper, lower and no child cries, respectively.

Figure 5.2: Blind-nanny-small: There are only 3 observations Z = {lower-child-crying, upper-child-crying, failed} and four actions $A = \{u, d, l, r\}$. Synthesized FSC is best in a family of all 2-FSC concerning LRA minimization objective that some child cries.

216 possibilities creating design-space. There are two LRA objectives specifying the time spent with each child.

- Smaller version of *blind-nanny* problem is *blind-nanny-small* (Figure 5.2a). Studied LRA property is how much time at least one child cries.
- *Endless-maze* is another grid-like example inspired by [6]. An agent is wandering in a maze and is teleported to a random location once he finds a goal state. The objective is to reach the goal state as fast as possible an infinite number of times.
- In *Drone-4-1* model, the goal is to find a controller of a drone maximizing the LRA probability to be in a given goal state in a grid map. Additionally, it is trying to avoid an agent, which is moving around the grid in a probabilistic manner.
- *Crypt4* is modified dining cryptographers problem [12]. Four cryptographers gather around a table and communicate using an anonymous recipient algorithm forever. The design space of this model is enormous as it consists of 2068 states.

5.2 Impact of used FSC memory for POMDPs with LRA objectives

The search for FSC implementing a solution to the POMDP problem starts with a memoryless controller because it takes the least time and some problem solutions do not require memory nodes. Memory nodes are incrementally added in order to find better solutions, this strategy assures that small good FSCs are found first. When the entire k-FSC family is pruned and the optimal value equals the optimal value found in the k-1 family, then an algorithm achieved solution bound, and adding more memory will not improve the optimum. The following investigation of selected examples will demonstrate this point.

Small-blind-nanny is the first investigated problem, described in the Figure 5.2. The objective is to calm the child as soon as possible when it starts to cry. Each child stops crying immediately after reaching its location. In the family of 1-FSC, it is not a problem to

k-FSC	optimum memory-demo	blind-nanny-sm
1	0	1
2	0.0095	0.074
3	0.026	**0.04**
4	0.05	0.04
5	0.077	0.04
6	0.095	0.04
7	**0.103**	0.04
8	0.103	0.04

Figure 5.3: Eventually, adding more memory nodes will not improve the optimum. Bold values mark solution values. For *memory-demo*, adding more than seven memory nodes does not improve the optimum and only three memory nodes are required to find a solution to *blind-nanny-sm* problem. Both require at least one memory node.

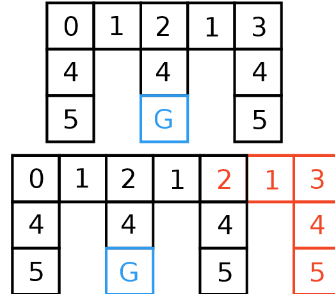


Figure 5.4: Endless-maze (top): The agent searches for the goal state (G) and is teleported to a random location each time it's found. Actions $A = \{u, d, l, r\}$. States colored red extends model creating *endless-maze-large* (bottom). Numbers label observation in each state. The objective is to find a goal always as soon as possible.

reach any of the children, but it is not possible to return back to the other one, thus, in the long run, at least one child will always cry. However, adding one memory node allows the nanny to determine her position and stay near both children. The best 2-FSC is outlined in 5.2b. No child is crying at the begging, so the objective is to reach the right bottom corner – the action to take is *right* if the agent is at the first child and *down* if it is at the upper child. Then, in the case that, e.g., the upper child starts to cry, the nanny's observation changes to *UC*, she takes the action *up*, and the memory changes to 1 in order to note her location. That child calms down and she returns to the right bottom location and notes it by changing memory back to 0. A similar approach is taken when the bottom child starts to cry. Incidentally, there is also a *both-crying* observation, but it is never observed when the optimal strategy is followed, thus it is omitted from FSC for simplicity. The optimum can be improved even more by adding another memory node, as described in Figure 5.3. Using three memory nodes, the LRA that some child is crying is 0.04, and adding a fourth memory node does not improve the optimum anymore.

Memory-demo (recall Example 7) is a model designed to show how adding memory nodes improves LRA optimum. Eventually, it happens, that adding more memory does not improve the optimum, because it is more advantageous to take (risky) action b to reach the target state. Such an equilibrium state is achieved when 7 memory nodes are added because adding eight one does not improve the LRA objective to be in the target state. Monotonous improvement of the optimum value can be observed in Table 5.3 until the solution value of 0.103 is reached.

Endless-maze is depicted in Figure 5.4. Here, two memory nodes are required to 1) distinguish between two states having observation *1* and 2) choose different action at observation *4* to go down when the goal state is directly below and up otherwise. Adding more than two memory nodes does not improve optimum. However, in the extended version *endless-maze-large*, more memory nodes are useful. E.g., in the path from the right top corner to the goal state, the agent must remember to take the second (not first) turn down.

problem name	specification	fastest			best		
		time	optimum		time	optimum	
blind-nanny-small	LRA_{min}	<1	0.074	(2)	6	0.04	(3)
memory-demo	LRA_{max}	<1	0.077	(5)	43	0.103	(7)
endless-maze	LRA_{max}				<1	0.122	(2)
endless-maze-large	LRA_{max}	<1	0.071	(2)	TO/263	0.094	(4)
robot-battery	LRA_{max}	<1	0.133	(2)	TO/1001	0.214	(3)
drone-4-1	LRA_{max}	<1	0.044	(1)	1.43	0.135	(1)
crypt4	LRA_{max}	<1	0.037	(1)	TO	–	(1)

Figure 5.5: Overview on optimum values found by the AR method for various problems. Found optimum values and a number of k required memory nodes – marked (k) – are recorded. If any interesting value is found faster than the best-found value, we record it in the *fastest* column. Times are in seconds, less than one second is marked as <1. Hyphen (–) marks that no better value was found in a 1-hour timeout (marked as TO). When TO/time is specified, the optimum value is best only in an incomplete k-FSC search.

The objective is to maximize the LRA probability to be in the goal state. The solution strategy (2-FSC) maximizing LRA to 0.122 was found for the default model. And best value of 0.094 was found for the large model in a family of 4-FSCs. Considering larger state-space in a larger model, solutions are almost equally effective.

Q2: **Summary of adding memory nodes to selected examples.**
In the previous examples, it was discussed how adding memory optimizes LRA objectives. The optimization of optimum values by adding memory for *blind-nanny-small* and *memory-demo* are in Figure 5.3. The required number of memory nodes to find a solution is 3, 7, and 2 for *blind-nanny-small*, *memory-demo*, and *endless-maze*, respectively. The synthesis of FSCs wrt. optimization LRA specification using the AR method is summarized in Figure 5.5. In all cases, interesting results are found under one second, because the strategy searches in the smallest FSCs first. In the *endless-maze-large* and *robot-battery*, design-space explodes by adding more memory to already large problems, creating enormous families whose exploration takes more than one hour. In those cases, only part of the k-FSC family is pruned and the best-found solution is recorded. The last problem, *crypt4*, is large enough, that not even a family of 1-FSCs is pruned. The optimal value is found under one second and no better solution is found in the remaining time.

5.3 Evaluating AR performance regarding LRA properties

A synthesis was run on introduced models to investigate how much is the AR method effective in comparison with the baseline one-by-one method. Therefore, the synthesis was run using both methods on identical models and specifications. Some specifications consist of two properties, however, their values are the same for simplicity. In addition to LRA specifications, there are also reachability specifications in order to compare the acceleration of the AR method concerning different types of specifications.

The statistics about benchmark examples with various memory sizes are in Figure 5.6. It is evident that adding memory increases the design-space enormously. The one-by-one method implements the consistent scheduler enumeration and therefore, the design-space is the exact number of required iterations. The size of the quotient MDP describes the number

problem name (m)	design-space	quotient MDP size	avg. MDP size
robot-battery	$5 * 10^2$	65	58
robot-battery (2)	$2 * 10^{12}$	128	108
robot-battery-stay	$2 * 10^4$	71	47
storm-problem (4)	$2 * 10^4$	9	7
storm-problem (5)	$5 * 10^5$	11	8
blind nanny	$2 * 10^7$	27	19
endless-maze (2)	$3 * 10^8$	18	15
endless-maze-large (2)	$2 * 10^9$	27	23
drone-4-1	$2 * 10^{2}9$	1225	–
crypt4	$1 * 10^{122}$	2068	–

Figure 5.6: Comparison of design-spaces and quotient MDPs of the benchmark experiments. A number in parenthesis represents the number of used memory nodes, otherwise a memoryless controller is used. The AR method never terminated when synthesizing the crypt4 and drone-4-1 models, therefore, the avg. MDP size statistics are not available.

of states in an underlying system, influencing the speed of model-checking and thus also the synthesis speed. In the AR method, the splitting makes many states unreachable, making the size of the subfamily's quotient MDP smaller. The average MDP size outlines how many states became unreachable, by specifying the average number of states in model-checked MDP. Note that in trivial synthesis problems, where it is only necessary to do 1 iteration, it holds that quotient MDP size equals average MDP size. The average MDP size in non-trivial cases in the below tables was averaged once more to create an average MDP size record in the mentioned table.

Because changing the λ value changes the difficulty of synthesis using the AR method (recall Example 6), the synthesis is run using different λ values to demonstrate the extent to which synthesis time increases. The feasibility threshold is marked as λ_f. Synthesis difficulty is also highly connected to the number of iterations, so the number of iterations is next to the synthesis time in the *iters* column for the AR method. On the other hand, intuitively, changing the λ does not affect the synthesis time of the one-by-one method. In the below tables, the synthesis time unit is one second. Milliseconds are sometimes omitted to simplify tables. Please, refer to the attached medium for exact synthesis times or synthesis results. Timeout was set to 1 hour, so if the solution was not found in that period, it is marked as >3600. Sometimes, the <1 mark is used when synthesis was faster than one second.

	one-by-one	AR	
λ	time	time	iters
0.16	>3600	>3600	>860676
0.17	>3600	62	46849
0.20	>3600	34	25627
0.22	>3600	4	3681
0.25	>3600	<1	15

Figure 5.7: Robot-battery memory 2: LRA>λ [„exploring“], $\lambda_f = 0.1\overline{6}$

Robot-battery. Since the design-space size of the model is only 512, the solution of the synthesis of such a small problem is found in dozens of milliseconds even when using the one-by-one method. Therefore, the focus will be on synthesizing FSC with 2 memory nodes. In this case, the design-space is enormous and the one-by-one method is not feasible as it would take days to find a solution. The AR method is able to find the solution to a maximizing optimality problem in 78 seconds. The optimum value LRA_{max} is $0.1\overline{6}$ and, as Table 5.7 shows, the AR is not able to find the solution to the feasible synthesis problem. Specifically, when the λ value is 0.16, it is not able to find the solution in a one-hour timeout interval although it does 860 676 iterations. Increasing the value to 0.17 creates an unfeasible synthesis problem and the AR successfully terminates in about one minute. The quotient MDP of the entire family over-approximates maximum LRA to value around 0.25. When the λ value approaches that threshold, the solution is found in under one second because it requires only a few iterations.

λ	one-by-one time	AR time	AR iters
0.01	3.83	0.05	123
0.08	3.91	0.03	71
0.09	3.87	0.01	19
0.15	3.88	0.01	16
0.2	3.86	0.01	10

(a) LRA $> \lambda$ [„exploring“], $\lambda_f = 0.08\overline{3}$

λ	one-by-one time	AR time	AR iters
0.2	3.66	0.01	1
0.4	3.63	0.03	151
0.6	3.72	0.05	159
0.95	3.65	0.09	69

(b) P $> \lambda$ [„exploring“], $\lambda_f = 1$

Figure 5.8: Robot-battery-stay: Comparison of methods on liveliness LRA and liveliness reachability specification.

Robot-battery-stay. The design-space of this model is moderate, but it is enough to show the capabilities of the AR method. While the one-by-one method always requires 19 683 iterations to model check each realization of the family, the AR method needs at most 159 iterations. It is clear, that the synthesis time using the one-by-one method was always approximately 3.8 seconds. On the other hand, the AR method is able to find a solution to every problem in under 0.1 seconds. An interesting remark can be seen in the synthesis of the liveliness LRA property (Table 5.8a). The number of iterations increases from 71 to only 19 when the feasibility threshold λ_f is exceeded – λ is changed from 0.9 to 0.8. Although, the change did not greatly manifest in synthesis time. In general, the difficulty of synthesis keeps increasing while the λ is decreasing even though λ is moving away from λ_f.

As the model-checking reachability property is faster, the execution time of the one-by-one method is slightly smaller in Table 5.8b than 5.8a. When switching from liveliness to a safety property in the AR, the durations don't show a significant difference. The AR method performs at least 37 times better than one-by-one and acceleration regarding LRA and reachability properties is comparable.

λ	one-by-one time	AR time	AR iters
0.01	1.42	0.96	4672
0.22	1.4	0.92	4536
0.3	1.47	0.73	3626
0.49	1.42	0.08	352

(a) liveliness property, memory=4

λ	one-by-one time	AR time	AR iters
0.01	45	28	125836
0.22	44	25	117630
0.3	45	19	88014
0.49	45	2	7918

(b) liveliness property, memory=5

Figure 5.9: storm-problem: Comparison of synthesis speed of 4-FSCs and 5-FSC. The optimal value λ_f=0.5 is the same in both cases.

Storm-problem. The design-space of this model is comparable to the previous – *Robot-battery-stay* – model, however, the size of the underlying system is about 6 times smaller. Therefore, the synthesis is more than 2 times faster in the one-by-one method. The AR method does perform noticeably better than the one-by-one, as the synthesis time is at least 1.4 times smaller. Nevertheless, this is not as much of a speedup as in the previous examples. The performance is not different when comparing liveliness and safety property.

This model is evaluated with memory 4 and 5, even though adding memory does not change the feasibility threshold λ_f. The interesting note is, that the acceleration ratio of the AR method is similar when the memory node is added. E.g., the worst recorded acceleration is 1.4 and 1.6 in 5.9a and 5.9b, respectively. The best case scenario, for non-trivial synthesis, is speedup up to 23 times.

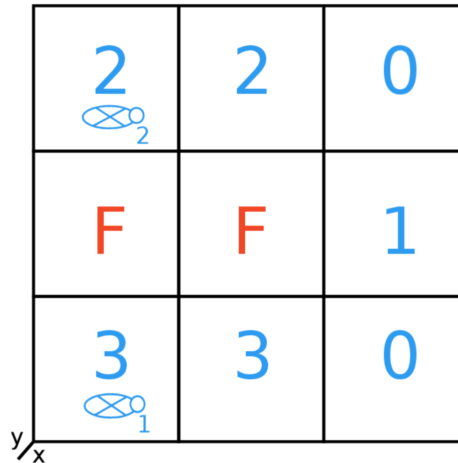


Figure 5.10: *Blind-nanny* problem: Blind nanny is supposed to take care of two children, however, she does only view adjacent fatal states, marked as red F. The direction of viewing distinguishes observations therefore there are 4 (blue) observations – lower, left, upper adjacent state is fatal and no adjacent state is fatal. Additionally, there are 3 observations deciding which baby needs care – the first or second baby is crying and no baby is crying. A combination of stated observations creates $4 \cdot 3 = 12$ total observations.

λ	one-by-one	AR
0.01	2872	10398
0.1	2884	10327
0.3	2882	9842
0.4	2945	3352
0.6	2874	2982
0.64	2951	2544
0.65	2971	569
0.66	2941	<1

Figure 5.11: Blind-nanny: The AR method does not always outperform the one-by-one on synthesis regarding LRA properties. Two liveliness LRA properties with the same λ value.

λ	one-by-one	AR live	AR safety
0.05	2045	329	734
0.1	2061	281	683
0.3	2050	250	650
0.95	2058	109	589
0.98	2092	102	576

Figure 5.12: Blind-nanny: Comparison of methods on reachability properties – same threshold λ is used on both properties. Both liveliness (live) and safety properties are concerned.

Blind-nanny. This model has quite a large design space even without memory nodes, but the quotient MDP size is moderate. It is the special model because the AR method does not outperform the one-by-one method in all cases, as stated in 5.11. Specifically, the one-by-one method terminates in under 3000 seconds every time, in contrast with the AR method, which takes more than 10k seconds in the hardest synthesis problems. When the λ value is 0.66, it is necessary to do only a few refinements of the quotient MDP to come across the over-approximation threshold around 0.65, meaning that algorithm terminates after a few iterations. Lowering the λ only by 0.02 already creates a challenging synthesis problem, but the AR still remains faster than one-by-one. However, the AR does not find the solution faster, when the λ decreases to 0.6 and below. The feasibility threshold λ_f is around 0.35, using the same λ for both liveliness properties. It is evident, that the execution time of the AR method increases almost 3 times when the feasibility threshold is crossed. Additionally, the synthesis wrt. to only one LRA property was run, to make sure that it is not multiple-objectives ruining it for the AR method.

However, the synthesis with respect to reachability properties (5.12) turned out differently. Here, the AR is at least 6 times faster for liveliness and around 3 times faster for safety property. In this case, the feasibility thresholds are 0 and 1 for safety and liveliness properties, respectively, meaning that all problems have feasible solutions.

Q3: **To summarize, the AR method performs multiple times better in most of the examples.** In the introduced examples, the synthesis using the AR is faster in three out of four synthesis problems. Other synthesis problems are summarized in Figure 5.13. As stated earlier, λ value has a significant impact on the execution time of the AR method, hence the synthesis was run using various λ values to create the easiest and the hardest synthesis problems. Both methods are slightly modified to prune an entire design-space instead of returning the first satisfying assignment. That allows to compare the methods without the factor of luck. Synthesis problems with a tiny design-space were equipped with memory nodes because otherwise the synthesis using both methods is faster than one second and the comparison is misleading. Returning to the table, it is evident, that the range of acceleration differs significantly. E.g., for the *storm-problem*, in the worst case, the acceleration was only 1.5, but when the synthesis problem was easier, it was up to 25. In the *robot-battery* problem, the one-by-one method is not feasible at all and the AR method is able to find a solution only when the synthesis is unfeasible. There are multiple

problem name (m)	LRA AR speedup worst	LRA AR speedup best	P AR speedup worst	P AR speedup best
storm-problem (4)	1.5	25	17	18
robot-battery (2)	TO	∞	–	–
robot-battery-stay	52	370	120	364
blind-nanny	0.277	5	3	20
blind-nanny-small (2)	4	130	$1.8 \cdot 10^3$	$3.5 \cdot 10^3$
endless-maze (2)	$4.4 \cdot 10^3$ *	$2 \cdot 10^4$ *	325 *	515 *
endless-maze-large (2)	$1.3 \cdot 10^4$ *	$2 \cdot 10^5$ *	$2 \cdot 10^3$ *	$5 \cdot 10^5$ *
crypt4		TO		TO

Figure 5.13: Comparing speedup between the AR method and one-by-one method on various models. The comparison is on both LRA and reachability (P) specifications. To provide an acceleration range, there are the best and worst recorded values. Small models are concerned with more than one memory node – marked as (m). Hyphen means not tested and TO means one-hour timeout. When the AR terminated in the timeout and one-by-one did not, the estimated time to prune the entire design-space (using the one-by-one method) was used. The * marks these cases.

cases when the one-by-one method was not able to prune the entire design-space until the timeout. However, because the method's run-time is linear, even a brief period of time can provide an estimate of when it will finish. In that case, the estimate is used to calculate the acceleration. It was not possible to solve the *crypt4* problem even using the AR method. For the *blind-nanny*, the AR method was at most 5 times better, however, in most cases, the one-by-one execution took up to 3 times less time. In the next section, this problematic model will be the subject of a detailed review in order to find out, why is the synthesis using the AR method slower.

Q4: **The synthesis with reachability specifications shows comparable results in the best-case scenarios.** The best case scenario speedup differs significantly only in *blind-nanny-small* problem – where it is faster using reachability specifications – and *endless-maze* where it is faster using LRA specifications. However, the **worst case accelerations between LRA and reachability properties differ**. The execution time with the reachability specifications doesn't decrease that much as with LRA. In fact, the reachability acceleration decreases more only in the *endless-maze-large*. In all other cases, the speedup decreases significantly less. E.g., in *storm-problem*, for LRA the acceleration decreases from 25 to a mere 1.5 while for reachability, it decreases only from 18 to 17. Additionally, when reachability specification is concerned, the AR always outperforms the one-by-one method even on the peculiar *blind-nanny* problem.

5.4 Investigating limitations of the AR method regarding LRA properties

While the abstraction-refinement method is faster in most cases, there is one problematic example where one-by-one enumeration performs better. For a better understanding of why this is happening, various ideas are examined to reveal details about the model. The first idea is, if a model-checking time of a larger family is greater than for smaller, refined, families – one would expect, that the model-checking time of the quotient MDP of an entire

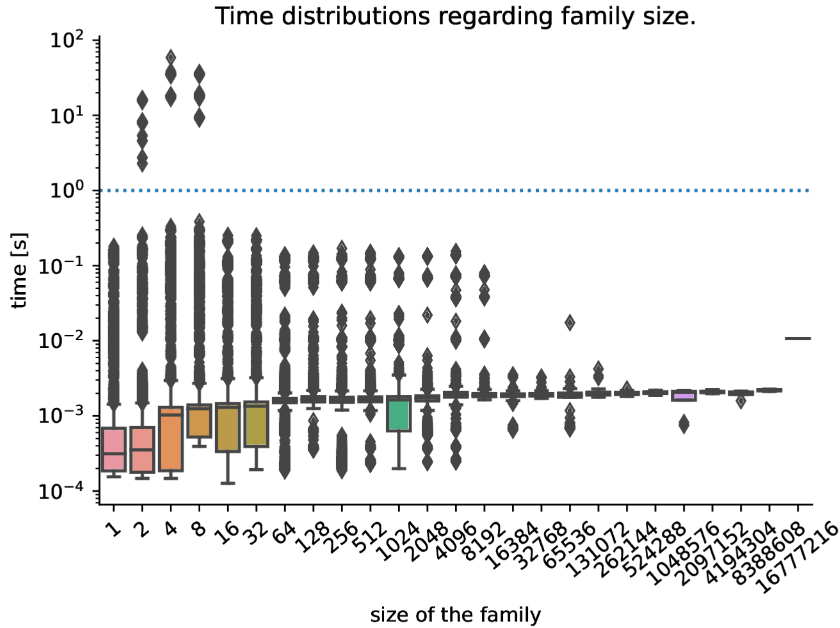


Figure 5.14: Distributions of model-checking times in seconds with respect to different family sizes. It is clear, that analysis time is not significantly larger when the family size increases.

family would be the largest. The second question is: how much does the number of states in the MDP affect model-checking time? Lastly, for MDPs, the Long-Run Average property is computed using iterative algorithms, which rely on iterating as many times as it is needed for the algorithm to converge. Therefore, it will be examined, how much is the converge time affected by small transition probabilities.

Does large design-space impact model-checking time?
In most of the cases, it holds, that the number of states in quotient MDP decreases with a number of refinements, i.e., refined families have smaller quotient MDPs in comparison with their super quotient. Therefore, it is reasonable to consider, that model-checking the quotient MDP of larger families does take more time in general. To verify this idea, the model-checking time and design-space of each quotient MDP was noted during the synthesis of *blind-nanny* model. The used λ value was 0.6, as it is the first value, where is one-by-one faster in Table 5.11.

The records were used to produce graph 5.14. There is a boxplot for each size of the family, to show the distribution of model-checking times for the corresponding family size. It is evident, that model-checking MDPs representing larger design-space are slightly larger than model-checking refined MDPs. However, the **difference is insignificant**, as it is not even one order of magnitude. On the other hand, it is clear, that outline members have extreme values – more than 10^4 times larger than mean values.

The total number of records is 434 440 and only 504 out of them are over one second, in 5.14, they are separated by dotted line. Surprisingly, the analysis times of that small subset of quotient MDPs sum up to 9636.3 seconds out of a total time of 11087 seconds. Therefore, around 0.116% of model-checking calls took up to 87% out of total model-checking time. If it would be possible to reduce the model-checking time of those extreme values, then the

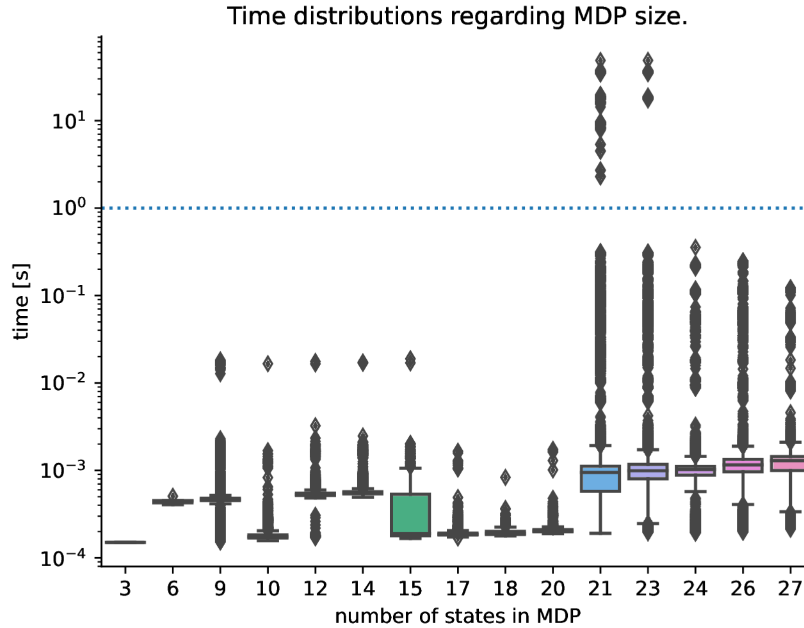


Figure 5.15: Graph showing distributions of model-checking times in seconds with respect to the quotient MDP size (state-space). It is obvious that the size of an MDP does affect analysis time. While the majority of times is under 10^{-2} there are extreme values reaching almost a minute.

synthesis of the entire family would be many times faster, and, more importantly, the AR method would be able to outperform the one-by-one synthesis.

Does large state space impact model-checking time?
Naturally, the difficulty of a model-checking task scales with a number of states in a provided model. In the *blind-nanny* problem, the state-space is not huge and therefore, it is a question, of whether or not large MDPs have an impact on the AR performance. During the same synthesis run of the blind-nanny model, as was already motioned, the sizes of MDPs were noted too.

The graph in Figure 5.15 shows the distributions of synthesis times with respect to different numbers of states in particular quotient MDP. It can be observed, that model-checking is more time-consuming in the models having more than twenty states and in general, there is an incline as the number of states increases. Specifically, the mean model-checking time increases 10 times when comparing the lowest and the highest number of states. The majority of the time-consuming outliners are on models with more than twenty states. In summary, larger state spaces do **have distinct impact** on model checking time. However, the extreme values are still the biggest concern, which is again separated by a dotted line as in Figure 5.14.

How much do small transition probabilities increase converge time?
Iterative algorithms computing LRA properties on MDPs do heavily rely on many iterations till the precise solution is found. Furthermore, at each iteration, the transition values are used to approach a solution. Therefore, they have a direct impact on the converge time.

In the examined model, there is a variable determining the probability that any of the children start to cry. In order to change transition values in the model, the variable will

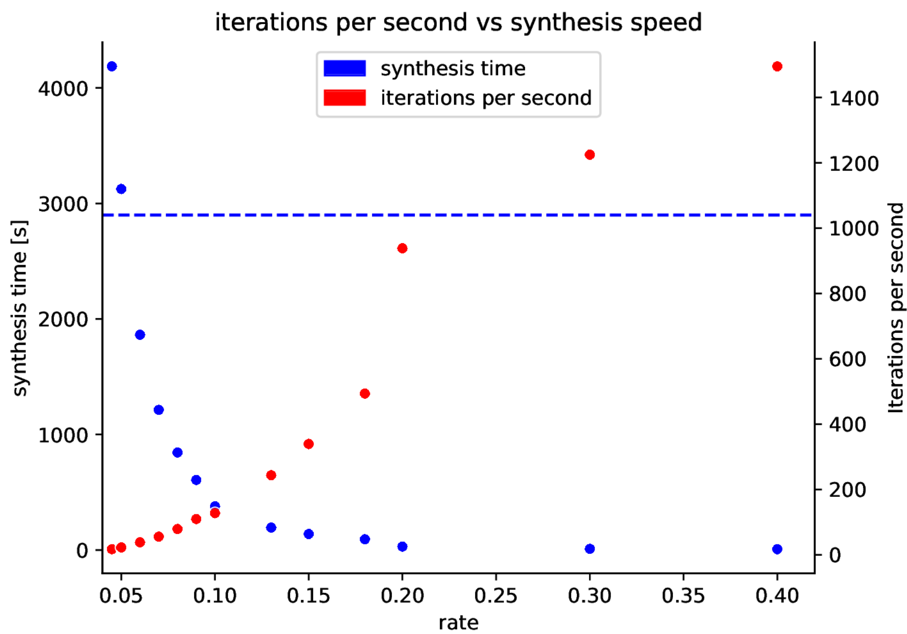


Figure 5.16: Impact of small transition probabilities on synthesis speed – the number of model-checks per second – using the AR method. Rate is variable in *blind-nanny* problem, indicating the probability that some child starts crying. The dashed line represents the average synthesis speed using the one-by-one method,

be modified. However, note that the variable is not an exact transition value, because there are also other constants, which must be multiplied by the rate in order to create the actual transition value of an underlying MDP. Therefore, the transition probabilities are even smaller.

The synthesis was run multiple times and the rate was increased in each iteration. It was initially set to a pretty low value (0.05) where was one-by-one enumeration faster. However, as the graph 5.16 shows, increasing the variable to 0.06 shortens the synthesis time by one-third, making the AR method faster. Synthesis speed decays exponentially with the increasing rate. Increasing the rate has a significant impact on the solution because the number of iterations decreases with it. Therefore, the synthesis time is not sufficient to show the impact on small transition probabilities. For this purpose, there is the number of iterations per second in the chart. It is clear, that lower rates have **huge impact** on converge time, as the number of iterations per second drastically decreases.

The same experiment was carried out for the one-by-one method to prove, that changing transition values doesn't have any impact on the synthesis speed, because steady-state distribution is obtained by solving a system of linear equations, rather than using an iterative approach. In the graph 5.16, there is the dashed line, representing the synthesis speed of the one-by-one method. Additionally, the one-by-one method was able to prune around 6000 realizations per second.

Q5: **Optimization propositions based on performed analysis.**
The analysis of the problematic example indicates, that the majority of the synthesis time takes model-checking of a tiny subset of families. Additionally, each family of that subset represents a maximum of 8 realizations (MCs) and small transition probabilities do not impact synthesis using the one-by-one method (model checking MCs). Therefore, if there was a way to predict, that the model-checking time of a concrete quotient MDP would take a long time, then it would be reasonable to let the one-by-one method synthesize that sub-family. This approach would preserve the synthesis speed of the AR method while the outliners would be eliminated. In the presented example, there are around 500 extreme values, if all of them would represent a model-checking family of size 8 then it would be $500 * 8 * 10^{-3} = 4$ seconds, instead of 9636 seconds.

However, predicting if the model-checking of concrete MDP would take more time than iterating each realization is not trivial. The prediction could be based on the distribution of transition values in the model and the size of the family it represents. Further analysis is required to obtain details distinguishing problematic models. For the examined example, it could be also interesting to see, if using the one-by-one methods on all families of size 8 and smaller would be faster.

Another proposition is to start both synthesis methods for each (sub-)family on separated threads and terminate the slower method, once the other one finds a solution. This introduces threading overhead, but the fastest possible solution would be used, essentially eliminating extreme out-liner model-checking times.

Chapter 6

Conclusion

The aim of this work was the synthesis of finite-state controllers for partially observable Markov decision processes with respect to steady-state properties. It has been achieved with the utilization of the Abstraction-Refinement (AR) method. The AR method argues about the subset (family) of candidate FSC by creating the abstraction represented with an MDP. If the abstraction is too coarse, then the family is *refined* and two sub-families are created. The analysis of sub-families follows and this process is repeated until the satisfiable FSC is found. The principles of the counterexample-guided inductive synthesis method (CEGIS) were adapted to design a novel algorithm for generating counter-examples (CE) wrt. steady-state properties. It was shown, that the generation of CE is not possible inside an ergodic MC, because the CE is essentially a sub-system, and the creation of the sub-system changes drastically the behavior of system wrt. steady-state properties. Thus, the generation is focused on non-ergodic MCs.

The AR method was compared with a baseline one-by-one exploration algorithm on a specially designed benchmark. The benchmark consists of 8 POMDP problems written in the PRISM language with non-trivial steady-state properties. The AR methods outperformed the baseline algorithm in 7 out of 8 problems. The AR method was up to 10^5 times faster. The one problematic example was examined and it showed, that 504 out of 434440 model-check calls on MDPs took around 87% percent of the synthesis time. The low transition rates have been shown to be responsible for this effect. The AR method was used to analyze the benchmark models wrt. reachability properties as well. The nature of the AR method implies that synthesis speed depends on the values defining investigated properties. Therefore, many different values were examined on both types of properties. For steady-state properties, the synthesis times decreased significantly when the best and worst possible values were used. The one of largest recorded decreases was from 25 to a mere 1.5, while on this model with reachability properties, the synthesis time decreased only from 18 to 17. This shows that the analysis wrt. steady-state properties is more difficult, however, the method is still applicable to many problems.

Future work could be the implementation of the proposed CE-finding algorithm. That was not included in this work because the CE generation is implemented in the synthesis-fork of the STORM model-checker, rather than in PAYNT. Based on the performed analysis, the AR method could also be improved by occasionally switching to the one-by-one method during the execution. Determining the appropriate switching moment is the subject of future research. Additionally, future research could be based on developing effective strategies for smart splitting families by considering the algorithms computing the steady-state values on MDPs, to avoid the current AR limitations.

Bibliography

[1] Ahmadi, M., Sharan, R. and Burdick, J. W. Stochastic Finite State Control of POMDPs with LTL Specifications. *CoRR*. 2020, abs/2001.07679. Available at: https://arxiv.org/abs/2001.07679.

[2] Akshay, S., Bertrand, N., Haddad, S. and Hélouët, L. The Steady-State Control Problem for Markov Decision Processes. In: Joshi, K., Siegle, M., Stoelinga, M. and D'Argenio, P. R., ed. *Quantitative Evaluation of Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, p. 290–304. ISBN 978-3-642-40196-1.

[3] Andriushchenko, R. *Computer-Aided Synthesis of Probabilistic Models*. Brno, CZ, 2020. Master's thesis. Brno University of Technology, Faculty of Information Technology. Available at: https://www.fit.vut.cz/study/thesis/22997/.

[4] Andriushchenko, R., Češka, M., Junges, S. and Katoen, J.-P. Inductive Synthesis for Probabilistic Programs Reaches New Horizons. In: Groote, J. F. and Larsen, K. G., ed. *Tools and Algorithms for the Construction and Analysis of Systems*. Cham: Springer International Publishing, 2021, p. 191–209. ISBN 978-3-030-72016-2.

[5] Andriushchenko, R., Češka, M., Junges, S., Katoen, J.-P. and Stupinský, Š. PAYNT: A Tool for Inductive Synthesis of Probabilistic Programs. In: Silva, A. and Leino, K. R. M., ed. *Computer Aided Verification*. Cham: Springer International Publishing, 2021, p. 856–869. ISBN 978-3-030-81685-8.

[6] Andriushchenko, R., Češka, M., Junges, S. and Katoen, J.-P. Inductive Synthesis of Finite-State Controllers for POMDPs. In: *Conference on Uncertainty in Artificial Intelligence*. Proceedings of Machine Learning Research, 2022, vol. 180, no. 180, p. 85–95. Proceedings of Machine Learning Research. ISSN 2640-3498. Available at: https://www.fit.vut.cz/research/publication/12776.

[7] Ashok, P., Chatterjee, K., Daca, P., Křetínský, J. and Meggendorfer, T. Value Iteration for Long-Run Average Reward in Markov Decision Processes. In: Majumdar, R. and Kunčak, V., ed. *Computer Aided Verification*. Cham: Springer International Publishing, 2017, p. 201–221. ISBN 978-3-319-63387-9.

[8] Baier, C. and Katoen, J.-P. *Principles of Model Checking*. 1st ed. Cambridge, Massachusetts London, England: The MIT Press, 2008. ISBN 78-0-262-02649-9. Available at: https://is.ifmo.ru/books/_principles_of_model_checking.pdf.

[9] Braziunas, D. *POMDP solution methods: a survey.* 1st ed. Department of Computer Science, University of Toronto, april 2003. Available at: https://www.techfak.uni-bielefeld.de/~skopp/Lehre/STdKI_SS10/POMDP_solution.pdf.

[10] Češka, M., Jansen, N., Junges, S. and Katoen, J.-P. Shepherding Hordes of Markov Chains. In: Vojnar, T. and Zhang, L., ed. *Tools and Algorithms for the Construction and Analysis of Systems.* Cham: Springer International Publishing, 2019, p. 172–190. ISBN 978-3-030-17465-1.

[11] Chatterjee, K., Saona, R. and Ziliotto, B. Finite-Memory Strategies in POMDPs with Long-Run Average Objectives. *Mathematics of Operations Research.* INFORMS. 2021. DOI: 10.1287/moor.2020.1116. Available at: https://hal.science/hal-02268862.

[12] Chaum, D. The dining cryptographers problem: unconditional sender and recipient untraceability. *Journal of Cryptology.* 1st ed. Springer-Verlag New York, Inc. 1988, vol. 1, no. 1, p. 65–75. DOI: 10.1007/BF00206326.

[13] Dehnert, C., Junges, S., Katoen, J.-P. and Volk, M. A Storm is Coming: A Modern Probabilistic Model Checker. In: Majumdar, R. and Kunčak, V., ed. *Computer Aided Verification.* Cham: Springer International Publishing, 2017, p. 592–600. ISBN 978-3-319-63390-9.

[14] Kochenderfer, M. J., Wheeler, T. A. and Wray, K. H. *Algorithms for decision making.* 1st ed. Cambridge : Massachusetts Institute of Technology: The MIT Press, 2022. ISBN 9780262047012. Available at: https://algorithmsbook.com/files/dm.pdf.

[15] Křetínský, J. and Meggendorfer, T. Efficient Strategy Iteration for Mean Payoff in Markov Decision Processes. In: D'Souza, D. and Narayan Kumar, K., ed. *Automated Technology for Verification and Analysis.* Cham: Springer International Publishing, 2017, p. 380–399. ISBN 978-3-319-68167-2.

[16] Kwiatkowska, M., Norman, G. and Parker, D. Stochastic Model Checking. In: Bernardo, M. and Hillston, J., ed. *Formal Methods for Performance Evaluation: 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007, Bertinoro, Italy, May 28-June 2, 2007, Advanced Lectures.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, p. 220–270. DOI: 10.1007/978-3-540-72522-0_6. ISBN 978-3-540-72522-0. Available at: https://doi.org/10.1007/978-3-540-72522-0_6.

[17] Lauri, M., Hsu, D. and Pajarinen, J. Partially Observable Markov Decision Processes in Robotics: A Survey. *IEEE Transactions on Robotics.* Institute of Electrical and Electronics Engineers (IEEE). feb 2023, vol. 39, no. 1, p. 21–40. DOI: 10.1109/tro.2022.3200138. Available at: https://doi.org/10.1109%2Ftro.2022.3200138.

[18] Madani, O., Hanks, S. and Condon, A. On the Undecidability of Probabilistic Planning and Infinite-Horizon Partially Observable Markov Decision Problems. In: *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence.* USA: American Association for Artificial Intelligence, 1999, p. 541–548. AAAI '99/IAAI '99. ISBN 0262511061.

[19] Shani, G., Pineau, J. and Kaplow, R. A survey of point-based POMDP solvers. *Autonomous Agents and Multi-Agent ...* . july 2012, vol. 27. DOI: 10.1007/s10458-012-9200-2.

[20] Silver, D. and Veness, J. Monte-Carlo Planning in Large POMDPs. In: Lafferty, J., Williams, C., Shawe Taylor, J., Zemel, R. and Culotta, A., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2010, vol. 23. Available at: https://proceedings.neurips.cc/paper_files/paper/2010/file/edfbe1afcf9246bb0d40eb4d8027d90f-Paper.pdf.

[21] Stewart, W. J. Performance Modelling and Markov Chains. In: Bernardo, M. and Hillston, J., ed. *Formal Methods for Performance Evaluation: 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007, Bertinoro, Italy, May 28-June 2, 2007, Advanced Lectures*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, p. 1–33. DOI: 10.1007/978-3-540-72522-0_1. ISBN 978-3-540-72522-0. Available at: https://doi.org/10.1007/978-3-540-72522-0_1.

[22] Velasquez, A. Steady-State Policy Synthesis for Verifiable Control. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, p. 5653–5661. DOI: 10.24963/ijcai.2019/784. ISBN 978-0-9992411-4-1. Available at: https://doi.org/10.24963/ijcai.2019/784.