Journal of Artificial Intelligence Research 82 (2025) 433-469 Submitted 06/2024; published 02/2025

An Oracle-Guided Approach to Constrained Policy Synthesis Under Uncertainty

Roman Andriushchenko IANDRI@FIT.VUT.CZ
Milan Češka CESKAM@FIT.VUT.CZ
Filip Macák IMACAK@FIT.VUT.CZ
Brno University of Technology, Brno, Czech Republic

Sebastian Junges SEBASTIAN.JUNGES@RU.NL
Radboud University, Nijmegen, The Netherlands

Joost-Pieter Katoen KATOEN@CS.RWTH-AACHEN.DE
RWTH Aachen University, Aachen, Germany

Abstract

Dealing with aleatoric uncertainty is key in many domains involving sequential decision making, e.g., planning in AI, network protocols, and symbolic program synthesis. This paper presents a general-purpose model-based framework to obtain policies operating in uncertain environments in a fully automated manner. The new concept of coloured Markov Decision Processes (MDPs) enables a succinct representation of a wide range of synthesis problems. A coloured MDP describes a collection of possible policy configurations with their structural dependencies. The framework covers the synthesis of (a) programmatic policies from probabilistic program sketches and (b) finite-state controllers representing policies for partially observable MDPs (POMDPs), including decentralised POMDPs as well as constrained POMDPs. We show that all these synthesis problems can be cast as exploring memoryless policies in the corresponding coloured MDP. This exploration uses a symbiosis of two orthogonal techniques: abstraction refinement—using a novel refinement method—and counter-example generalisation. Our approach outperforms dedicated synthesis techniques on some problems and significantly improves an earlier version of this framework.

1. Introduction

Markov decision processes (MDPs) are a prominent model for sequential decision making under aleatoric uncertainty. For a given MDP, a standard task is the computation of a policy[1] that optimises some (un)discounted reward over an (in)finite horizon (Puterman, 1994). These tasks can easily be extended to consider undiscounted rewards over an indefinite horizon, i.e., to maximise the reward obtained until reaching a goal state or maximising the probability that some temporal logic specification is satisfied (Baier et al., 2018). A range of well-known results ensure that these policies can be efficiently computed (both in theory and in practice). A practical concern is that in sequential decision making there are multiple, potentially competing objectives (Hayes et al., 2023; de Nijs et al., 2021). A popular research direction is therefore to consider constrained MDPs (Altman, 1999). Here, a second objective is imposed: The expected secondary reward (often the cost) must be above (or below) some value. These models can be further generalised towards finding Pareto optimal

1. We sometimes refer to the policies as controllers since we usually represent them as finite-state controllers.

policies (Forejt, Kwiatkowska, & Parker, 2012) at the expense of a significant computational cost.

In this paper, we extend the perspective of constrained MDP policies: in addition to imposing constraints on the collected reward, we also constrain the shape or structure of the (finite-state) controller representing the policy, i.e., of the policy space. These constraints can encode aspects that are often captured by using partially observable MDPs, e.g., that specific features of the state space is not available to the agent, but also encode that an agent must explicitly communicate to obtain specific information (which requires decentralised partially observable MDPs).

The constraints are more flexible than what is expressible with the extensions of MDPs mentioned above. By explicitly formulating the constraints, we can consider algorithms that find provably (almost) optimal policies within the constrained policy space. Constraining the shape of controllers may also be used to restrict the policies' complexity, e.g., in terms of memory used. This option is helpful if the policy must be explained—or even executed—by a human. The latter observation has motivated work that limits the number of α-vectors in partially observable MDP (POMDP) policies (Ferrer-Mestres et al., 2021) or searches for decision-tree shaped policies (Vos & Verwer, 2023)[2]. We can encode symmetries that ensure that an agent behaves similarly in various settings, or that a specific costly action is not taken too frequently. In a similar spirit, we may already know that a good policy is best explained using some (potentially unobservable) statistics of the current system state that can be easily memorised, e.g., that a pong player must remember the direction of a ball, even if that is not immediately observable, or that a user should remember the password for the repeat-password check.

To integrate these kinds of "controller shape" constraints, we propose a generic framework for the policy synthesis problem for MDPs *given a (symbolic description of)* the entire policy *design space*. The analysis of any fixed policy within this design space is straightforward: The main challenge is to handle large design spaces, i.e., design spaces with lots of flexibility and options. Our approach is inspired by the *syntax-guided program synthesis* (Alur et al., 2015) paradigm in which the design space (of programs) is represented by a program sketch describing syntactic constraints on candidate programs. The *oracle-guided inductive synthesis* approach (Jha et al., 2010; Solar-Lezama et al., 2005) to program synthesis uses a learner to select a candidate program from the design space. An oracle answers whether the candidate meets the specification and crucially provides additional information if it does not. Various instances of inductive synthesis exist, including counter-example guided synthesis (Solar-Lezama et al., 2006), synthesis from input-output examples (Osera & Zdancewic, 2015) and inductive logical programming (Cropper & Dumančić, 2022). These approaches seek for a program that adequately generalises training examples. In this paper, we replace the training examples (i.e. data) with a formal specification describing the desired behaviour of a program.

1.1 Problem Statement

We consider the following *feasibility* problem: given an MDP, a specification, and a—possibly huge but finite—design space of policies, find a policy (if any) such that the induced Markov

2. Note that in deep RL, the policy definition is also restricted to a fixed architecture.

chain (MC) satisfies the specification and all imposed constraints that are used to define the design space. These constraints can limit the size of the policy or encode domain-specific knowledge as in program sketching (Solar-Lezama, 2013). Such *satisficing* policies meet the specification while fulfilling all given constraints. The *optimal* feasibility problem aims at finding a policy that satisfies the specification under all given constraints in an *optimal* manner. Evidently, this is a more challenging problem. Finally, we consider relaxed optimal feasibility. The *almost optimal* feasibility problem aims to find a policy that is almost optimal, achieving a result that is 5%, say, away from the optimal value.

1.2 Motivating Examples

We sketch three related but quite different synthesis problems. As we will show, all these problems can be rather efficiently solved using the techniques presented in this paper.

1.2.1 Herman's Protocol

Herman's protocol (Herman, 1990) is a well-studied randomised distributed algorithm for token passing among network stations. The key idea is that the algorithm is self-stabilizing: for any initial situation, it converges (fast) to a situation where all participating stations agree on who has the token. In particular, the algorithm considers a unidirectional ring of an odd number of network stations where all stations have to behave similarly—it is an anonymous network. Each station stores a single bit and can read the internal bit of one (say, counter-clockwise) neighbour. A station has the token if the two legible bits are both one. All stations execute an action in a lock-step fashion: Stations for which the two legible bits coincide update their bit based on the outcome of a coin flip. Other stations copy the bit of their neighbour. In the context of this paper, we can say that this is a partially specified policy that runs on every station: It remains open how to pick the bias of the coin flip and how to use memory to e.g. minimise the expected time until stabilisation. Kwiatkowska et al. (Kwiatkowska, Norman, & Parker, 2012) used a collection $\mathcal{M}$ of Markov chains (MCs) modelling different policies for the stations. Their analysis considered each instance separately and revealed which memoryless policies for Herman's protocol performs best. In a setting where the probabilities range over $0.1, 0.2, \ldots, 0.9$, this results in the analysis of nine different MCs. However, the policy space may also include finite state controllers that are not memoryless. Does the expected time until stabilisation reduce if we optimise over this extended policy space? We remark that the policy space quickly grows when allowing that the station policy accesses some internal memory: in every step, there are now 9·9 strategies for selecting the coin flip, and for each memory cell and coin flip outcome, the memory can be updated, yielding 2·2·2 possibilities. All in all, this one-bit extension results in 648 candidates. Indeed, if one allows stations to make decisions depending on the token bits, both the coin flips and the memory updates are multiplied by a factor 4, yielding $10,368$ different policies. We consider an improved variant of this protocol that leads to over 3 million different policies.

1.2.2 Dynamic Power Management

Dynamic Power Management (DPM) is a widely studied problem in the literature, see e.g. (Benini et al., 1999; Gerasimou et al., 2015). It processes requests that are generated

externally in random intervals and are stored in a request queue of limited capacity—if the queue is full, the incoming requests are lost. A service provider processes the requests from the queue. A power manager (PM) observes the queue size and selects an operating mode for the service provider. The modes have a different power consumption and there is a random latency before the mode is changed. Requests can only be processed in active mode and due to random errors, this processing is not successful with a small probability. The aim of this case study is to synthesise the PM that minimises the power consumption during the operation time (that is finite but random) while ensuring that the expected number of lost requests is below a given threshold. We do not synthesise the PM from scratch but assume a skeleton of the PM to be at our disposal. This blueprint represents the a-priori knowledge about the PM policy, such as its main control flow. It is modelled as a *program sketch* (Solar-Lezama, 2013) describing the high-level control flow while omitting certain details of the PM such as the observable size of the request queue to the PM, the mode that needs to take on a given (observable) queue size, and the queue capacity. We also consider a more complicated variant of the DPM problem that distinguishes low and high priority requests (and puts these in two separate queues). The sketch describes over 43 million candidate programs.

1.2.3 Meeting in a Grid Dec-POMDP

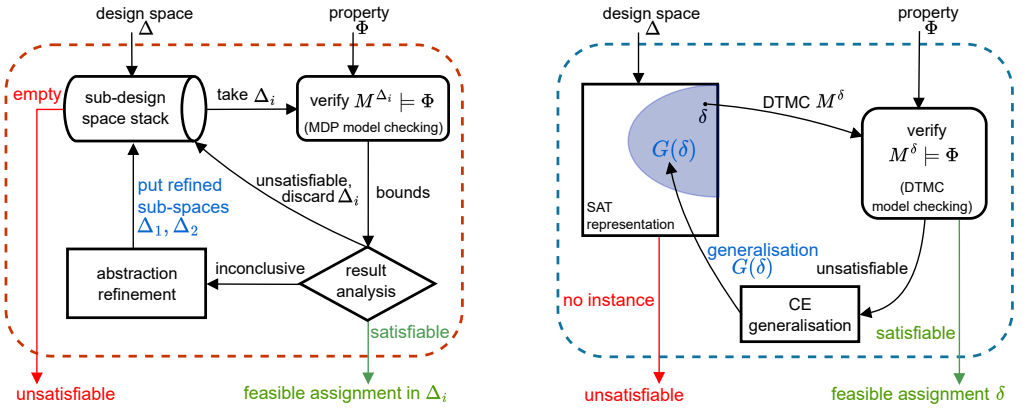
Finally, we consider finding a pair of agent policies, one for each of the two agents in a decentralised POMDP modelling a *meeting-grid* problem. This is a classical but challenging, decentralised planning problem (Dibangoye, Amato, & Doniec, 2012). Two agents, A and B, move on a grid with obstacles and the goal is to meet at some arbitrary cell in the grid. The agents only have partial information about their current position and as there is no communication between the agents, each agent solely relies on its own observation history. We consider each agent's policy to be represented as a finite-state deterministic controller with k memory nodes (k-FSCs) (Hansen, 1998). The goal is to find a pair of k-FSCs that minimises the expected number of steps until the agents meet. We consider different variants of the problem including a slippery grid (there is a probability that the agent fails to move) and different observation modes.

1.3 Algorithmic Approach

We now give an overview of our approach that is capable of tackling these synthesis problems.

1.3.1 Representation of the Problem

How do we represent these different synthesis problems in a uniform way? We coin the concept of *coloured* MDPs as succinct representations of the above synthesis problems. A coloured MDP describes a *finite* collection of possible *finite* policy configurations with their structural dependencies. This covers, amongst others, the synthesis of (a) programmatic policies from probabilistic program sketches, (b) possibly randomised finite-state controllers for (decentralised) partially observable MDPs (POMDPs), as well as (c) optimal finite-state controllers for constrained (PO)MDPs. Coloured MDPs can be seen as a symbolic manner of describing synthesis problems of various kinds in which a finite set of possible policies is given. We show that the synthesis problems (a) through (c) can be formulated as exploring



(a) Abstraction refinement (AR). (b) Counterexample generalisation (CEG).

Figure 1: Illustration of the main search strategies.

memoryless policies in their coloured MDP representation. More formally, the design space is assumed to be parameterised by a set K of discrete *parameters*. The design space then corresponds to a set of assignments $K \to V$ mapping parameters onto concrete values in V. Intuitively, these parameters can, e.g., encode what action to take on observing a green light. Applying a parameter assignment to a coloured MDP resolves all nondeterminism, thus yielding a finite-state Markov chain (MC). An analysis of this MC suffices to determine whether the parameter assignment—the corresponding policy—is satisficing. This can be done using off-the-shelf model-checking methods (Kwiatkowska, Norman, & Parker, 2011; Dehnert, Junges, Katoen, & Volk, 2017). The search for a satisficing policy can thus be considered as the search for a suitable parameter assignment. This view naturally fits within the realm of syntax-guided synthesis (Alur et al., 2015).

1.3.2 The Approach: Oracle-guided Synthesis

Tackling these synthesis problems consists of *exploring* the design space and *evaluating* the sets of candidate solutions. The efficiency of the exploration phase relies on a compact symbolic representation of the design space. The core of our approach consists of two orthogonal search strategies, each leveraging a different oracle: *Abstraction Refinement* (AR) and *CounterExample Generalisation* (CEG). These two approaches synergise through a tight integration.

Abstraction Refinement (AR) (illustrated in Fig. 1a (page 437) and described in detail in Sec. 3) starts off by representing the design space—consisting of all candidates—by a single coloured MDP. Action choices are used to model the various options in the design space. Only action choice combinations are allowed that result in an admissible candidate. As finding such consistent policies directly is practically infeasible, we use an *abstraction* of the design space to bound the possible values of the MCs in it. This is similar to the Q-MDP abstraction for finding bounds on POMDP policies (Littman, Cassandra, & Kaelbling, 1995). As refinement gives rise to multiple such abstract MDPs, it is convenient to consider a *set* of abstract sub-design spaces. One now selects an abstract sub-space Δ_i, creates an MDP M^{Δ_i} and sends it to an MDP verifier. Evaluating this abstract MDP serves as an oracle

yielding under- and over-approximations of the extremal probabilities in Δ_i w.r.t. satisfying the specification Φ. This oracle can be an MDP model checker (Kwiatkowska et al., 2011; Dehnert et al., 2017). If the under-approximation result satisfies Φ, AR finishes: all solutions in Δ_i comply with the specification. On the other hand, if the over-approximation result does not satisfy Φ, we can discard the sub-space Δ_i. In all other cases, the model-checking result is inconclusive, which means that it is possible that some candidates represented by Δ_i fulfill Φ, while others do not. That is, the abstraction is too coarse. Then a *refinement* splits Δ_i into appropriate sub-spaces. These sub-spaces are finer abstractions and thus more precise. The splitting strategy of candidate sets predominantly influences AR's efficiency. Thanks to the unifying representation using coloured MDPs, we can exploit (selection and) splitting strategies that apply to all our synthesis domains (i.e., (decentralised) POMDPs, sketches of probabilistic programs, ...).

The orthogonal search strategy **Counterexample Generalisation (CEG)** (illustrated in Fig. 1b on page 437 and described in detail in Sec. 4) operates on individual solutions. It thus considers MCs rather than (abstract) MDPs. From the (symbolically represented) set of individual solutions, it selects policy $\delta \in \Delta$, and discharges the induced MC M^δ to an off-the-shelf MC model checker. (Note that AR exploits analysing MDPs, whereas CEG uses MC model checking.) If the verification is affirmative, CEG stops and returns the found solution. Otherwise, for MC M^δ violating the specification Φ, it uses a *counterexample* (CE) obtained from the model checker. This CE is a critical sub-MC of M^δ that suffices to refute Φ. It is used to *generalise* δ to a set $G(\delta)$ of sub-designs that can be safely removed from the set of candidate solutions to be explored. The MC model checker, together with the CE generator, thus provides the alternative oracle. The smaller the CE, the larger the set of sub-designs that can be pruned. CEG thus attempts to find minimal CEs in the MC M^δ induced by δ. Phrased differently, a CE is preferably a minimal subset of parameter assignments in the coloured MDP.

The tight **integration** between AR and CEG (described in Sec.5) is key to the success of our approach. It builds on a refined notion of CEs defined for a given (sub-)space Δ using bounds on reachability probabilities over all MC M^δ with $\delta \in \Delta$. AR provides these bounds for particular sub-spaces and allows us to obtain smaller CEs, leading to better generalisation in the(sub-)space Δ. The integration is controlled by a manager that monitors the performance of the synthesis and adaptively switches between AR and CEG. The manager is also responsible for the result interpretation and reporting of the synthesis results.

1.3.3 Putting it Together

We briefly provide a high-level overview of the proposed approach (see Fig. 2). Our approach takes as input a concrete instance from the synthesis domain (upper left). This instance can either be a program sketch or (decentralised) POMDP given in the PRISM (Kwiatkowska et al., 2011) or Cassandra[3] format and automatically translates them into a coloured MDP. Other inputs include a specification—typically given as an optimisation objective—as well as some optional additional constraints, such as, e.g., an upper bound on the memory of the controller (lower left).

3. https://pomdp.org/code/pomdp-file-spec.html

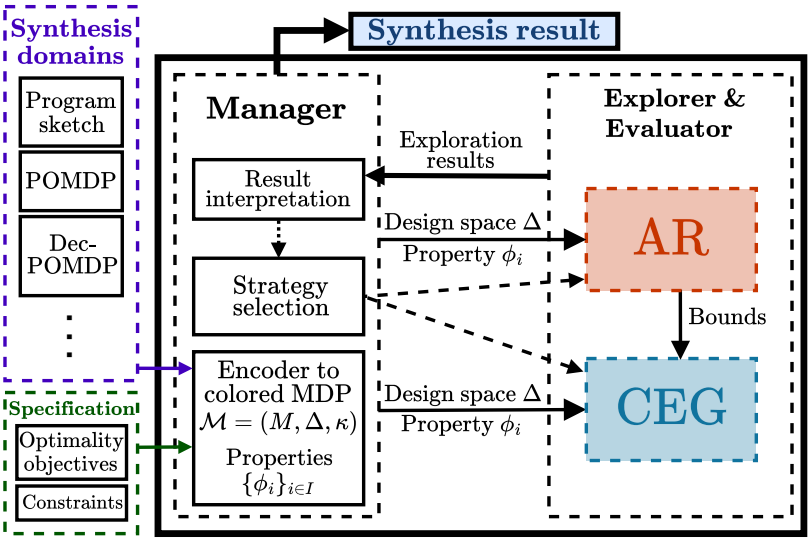


Figure 2: The overview of the proposed framework implemented in the tool PAYNT. The overview of abstraction refinement (AR) and counter-example generalisation (CEG) is showcased in Fig. 1a and Fig. 1b respectively.

Let us briefly illustrate the key steps of our approach on the two of the case studies from Sec. 1.2. We apply AR to the meeting-grid problem. In this case, the colouring of the MDP representation ensures that the partial observations of the agents are taken into consideration. The resulting policies are observation-based and decentralised, i.e., observations of one agent are not available to the other agent. When ignoring the colouring, AR yields policies that fully observe the state of both agents. A policy σ is inconsistent if it selects two different actions in states with the same observation. If σ is a maximising policy, our refinement strategy estimates which inconsistent selection has the highest impact on the upper bound and uses this to split the design space into two (or more) sub-problems.

Let us apply CEG on the DPM case study. Here, the MDP colouring ensures that the observations and the maximal queue size do not change and that the same service provider state is selected for equal occupancy levels. Assume CEG picks a solution where the queue capacity is 1. The CE obtained from model checking reveals that this decision already suffices to violate the upper bound on the expected number of lost requests. This means that even using the most energy-consuming power profile (no sleeping or idling) violates the given bound (basically, the queue is too small). This insight enables us to prune the design space significantly using the analysis of a single candidate solution only.

1.4 Main Contributions and Paper Structure

This paper presents:

- The concept of *coloured* MDPs to succinctly represent a *range of different synthesis problems*, including synthesising (a) programmatic policies from probabilistic program sketches, (b) finite-state controllers for (decentralised) partially observable MDPs (POMDPs), and (c) optimal controllers for constrained POMDPs.

- A *symbiotic* approach to synthesise policies, consisting of a *deductive* abstraction-refinement algorithm that uses a novel refinement strategy and an *inductive* counterexample generalisation technique.

- An *implementation* in the software tool PAYNT (Andriushchenko et al., 2021) that can handle uncertainty models with thousands of states and billions of possible policies in a fully automated manner.

- An *extensive experimental evaluation* using a wide set of benchmarks that cover the aforementioned synthesis problems. Our experiments reveal that PAYNT sometimes outperforms state-of-the-art, dedicated synthesis techniques and significantly improves upon (Andriushchenko et al., 2021).

We demonstrate that constraining the shape of the controllers representing the policies is essential in many problems relevant to both the model-checking and the planning community. Our approach builds on classical model-checking techniques such as abstraction-refinement and counterexample generalisation. The generic formulation presented in this paper, however, makes it straightforward to apply the approach to a wide range of planning and synthesis problems. Our empirical results show that this is competitive. As such, our work establishes further connections between probabilistic model-checking and probabilistic planning (Klauck et al., 2020).

This paper is based on the conference papers (Češka et al., 2019a; Andriushchenko et al., 2021, 2021) and unifies the presentation of those results. New ingredients are: the model of *coloured* MDPs to uniformly represent various synthesis problems, the *refinement strategy* in the abstraction-refinement oracle, a number of *new heuristics*, and improvement of the search strategy. Altogether, this yields significant improvements in PAYNT's performance, up to two to three orders of magnitude. Furthermore, the enriched experimental evaluation on a *larger* benchmark collection now also contains classical constrained POMDPs and Dec-POMDPs problems from the literature as well as a comparison with dedicated tools for those classical problems.

1.4.1 Structure of the Paper

Section 2 introduces *coloured* MDPs and formalises the problem statements. Section 3 and 4 present synthesis based on *abstraction refinement* and *counterexample generalisation*, respectively. Section 5 describes their symbiosis. Section 6 presents an experimental evaluation: i) it first demonstrates the capabilities of the proposed approach on three case studies from different application domains, ii) it compares the performance of our approach with state-of-the-art methods, and iii) it demonstrates the performance improvements with respect to the previous variants of the synthesis algorithms.

2. Problem Statement

This section presents the necessary preliminaries, introduces coloured MDPs, and formulates our problem statement(s).

2.1 Preliminaries

2.1.1 Distributions

A (discrete) *probability distribution* over a countable set A is a function $\mu\colon A \to [0,1]$ s.t. $\sum_a \mu(a)=1$. The set $\text{supp}(\mu) \coloneqq \{a \in A \mid \mu(a) > 0\}$ is the *support* of μ. The set $Distr(A)$ contains all distributions over A. We use Iverson bracket notation, where $[x] = 1$ if the Boolean expression x evaluates to true and $[x] = 0$ otherwise.

2.1.2 Markov Decision Processes (MDPs)

An *MDP* is a tuple $M = (S, s_0, Act, \mathcal{P})$ with a countable set S of states, an initial state $s_0 \in S$, a finite set Act of actions, and a partial transition function $\mathcal{P}\colon S \times Act \nrightarrow Distr(S)$.

Let $Act(s) \coloneqq \{\alpha \in Act \mid \mathcal{P}(s,\alpha) \neq \bot\}$ denote the set of actions available in state $s \in S$. We assume $Act(s) \neq \emptyset$ for each $s \in S$ (no deadlocks). An MDP with $|Act(s)| = 1$ for each $s \in S$ is a *Markov chain (MC)*. We denote $\mathcal{P}(s,\alpha,s') \coloneqq \mathcal{P}(s,\alpha)(s')$. State s is absorbing if $\text{supp}(\mathcal{P}(s,\alpha)) = \{s\}$ for all $\alpha \in Act$.

A (finite) *path* of an MDP M is a sequence $\pi = s_0\alpha_0 s_1\alpha_1 \dots s_n$ where $\mathcal{P}(s_i,\alpha_i,s_{i+1}) > 0$ for $0 \leq i < n$. We use $last(\pi)$ to denote the last state s_n of path π. Let $Paths^M$ denote the set of all finite paths of M.

A (deterministic, memoryless) *policy* (aka: scheduler) is a function $\sigma\colon S \to Act$. Let Σ^M denote the set of policies for MDP M. A policy $\sigma \in \Sigma^M$ induces the MC $M^\sigma = (S, \mathcal{P}^\sigma)$ where $\mathcal{P}^\sigma(s) = \mathcal{P}(s,\sigma(s))$.

2.1.3 Specifications

We primarily consider indefinite-horizon reachability or expected total reward properties (Puterman, 1994). Formally, let $M = (S, s_0, Act, \mathcal{P})$ be an MC, and let $T \subseteq S$ be a set of *target states*. Let $\mathbb{P}^M[s \models \Diamond T]$ denote the probability of reaching (some state in) T from state $s \in S$. If $\mathbb{P}^M[s \models \Diamond T] = 1$, then $\mathbb{E}^M[s \models \Diamond T]$ denotes the expected reward accumulated from s before reaching T. Let $\mathbb{P}^M[\Diamond T]$ and $\mathbb{E}^M[\Diamond T]$ denote $\mathbb{P}^M[s_0 \models \Diamond T]$ and $\mathbb{E}^M[s_0 \models \Diamond T]$, respectively. The superscript M is omitted if the MC is clear from the context. For MDPs, specifications are taken over the best and worst possible resolution of the non-determinism. Let MDP $M = (S, s_0, Act, \mathcal{P})$. The *maximal* reachability probability of T for state $s \in S$ in M is $\mathbb{P}^M_{\max}[s \models \Diamond T] \coloneqq \sup_{\sigma \in \Sigma^M} \mathbb{P}^{M^\sigma}[s \models \Diamond T]$. The maximum reward ($\mathbb{E}^M_{\max}$) specification and the minimum variants ($\mathbb{P}^M_{\min}$ and $\mathbb{E}^M_{\min}$) are defined analogously. Formal definitions can be found e.g. in (Baier et al., 2018).

Furthermore, we are interested in comparing the performance of our approach to existing methods for *discounted* (infinite-horizon) total reward objectives. Let $\gamma \in [0,1)$ be the *discount factor*. For a discounted total reward objective, we are interested in rewards collected along all infinite paths, i.e. we do not consider targets. In every step along a path, γ is multiplied to all future rewards as discounting: $\mathbb{E}^M\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$ is the expected discounted total reward for state s_0 in MC M with discount factor γ where random variable r_t denotes the reward collected at the t-th step t M. With discounting, the immediate reward is favoured over rewards collected later on a path.

Note that, under the assumption $\mathbb{P}^M_{\max}[s \models \Diamond T] = 1$, the undiscounted reachability reward specification is colloquially known as the stochastic shortest path (SSP) problem (Bertsekas,

2005). The discounting can be reduced to an SSP problem by introducing a fresh target state reachable via any action with probability $1-\gamma$. Generalised SSP (GSSP) extends SSP with reachability probability and other specifications, e.g. undiscounted total reward without target states (Kolobov et al., 2011). Our framework can be directly extended to GSSP problems.

2.2 Coloured MDPs

In this section, we introduce the concept of coloured MDPs which allows us to succinctly reason about different types of synthesis problems.

2.2.1 Parameter Assignments

We assume that the synthesis problem is given as an MDP equipped with a finite set H of *parameters* with $D(h) \subset \mathbb{N}$ denoting the finite domain of parameter $h \in H$. The parameters represent the design choices to be made. A *total assignment* δ of parameters maps each parameter to an element of its domain: $\delta\colon H \to \mathbb{N}$ where $\delta(h) \in D(h)$. It is assumed that δ induces an MC M^δ, i.e., (the behaviour of) the candidate solution given by δ is described by the MC M^δ.

Let Δ denote the set of all possible total assignments δ. This represents the *design space*. The assignments in Δ induce a set of MCs with the same state space but distinct topologies; i.e., certain states may be unreachable under one assignment but not under another.

See Fig. 3 (left) on page 443 illustrates such a set. The goal of the synthesis is then to find a specific total assignment $\delta \in \Delta$ of parameters such that M^δ satisfies the specification.

Assignment δ is *partial* if δ is a partial mapping, i.e., $\exists h \in H\colon \delta(h) = \bot \notin D(h)$. Let $\overline{\Delta}$ denote the set of all (partial and total) assignments. Assignments $\delta, \delta' \in \overline{\Delta}$ are *compatible*, denoted $\delta \sim \delta'$, if they agree on all parameters that are defined in both assignments: $\delta \sim \delta'$ iff $\forall h \in H : (\delta(h) \neq \bot \wedge \delta'(h) \neq \bot) \Rightarrow \delta(h) = \delta'(h)$.

2.2.2 Coloured MDPs

We now define *coloured* MDPs to compactly represent the set of induced MCs M^δ for each $\delta \in \Delta$.

Definition 1 (Coloured MDP) *A* coloured MDP *is a tuple* $\mathcal{M} = (M, \Delta, \kappa)$ *with MDP* M*, design space* Δ *of assignments of parameters* H*, and a* colouring $\kappa\colon S \times Act \to \overline{\Delta}$ *satisfying* $\forall \delta \in \Delta\, \forall s \in S\, \exists!\, \alpha \in Act(s)\colon \kappa(s,\alpha) \sim \delta$.

The colouring κ assigns to each action α in state $s \in S$ a (partial) parameter assignment $\kappa(s,\alpha)$ that this action is associated with. Given assignment $\delta \in \overline{\Delta}$, let $\mathcal{M}^\delta$ denote the induced MDP containing only actions that are compatible with δ, i.e., $\mathcal{M}^\delta := (S, s_0, Act, \mathcal{P}^\delta)$ where $\mathcal{P}^\delta(s,\alpha) = \mathcal{P}(s,\alpha)$ if $\kappa(s,\alpha) \sim \delta$ and $\mathcal{P}^\delta(s,\alpha) = \bot$ otherwise. The condition on the colouring in Def. 1 ensures that if δ is total, then in each state $s \in S$ there is exactly one action available, i.e. $\mathcal{M}^\delta$ is an MC that equals M^δ. If δ is partial, in each state of MDP $\mathcal{M}^\delta$, at least one action is available, i.e., $\mathcal{M}^\delta$ contains no deadlocks.

Example 1 *Fig. 3 illustrates the concept of coloured MDPs. Assume a synthesis problem given by parameters* h_1 *and* h_2 *with domains* $D(h_1) = D(h_2) = \{0,1\}$*. The design space thus*

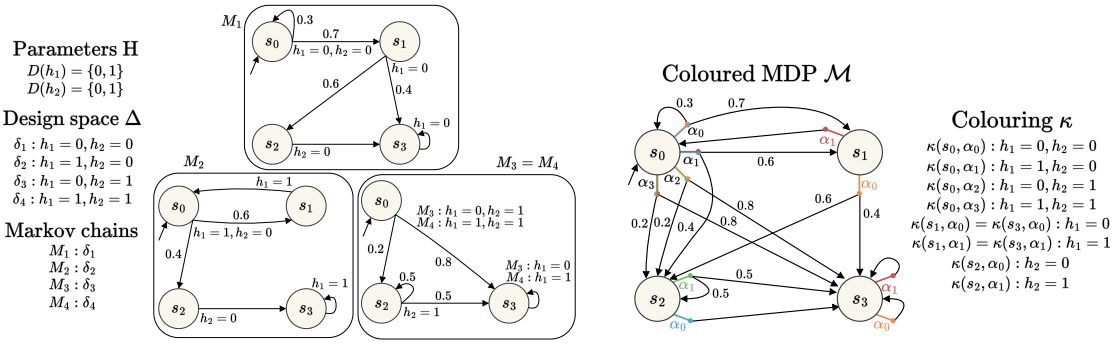


Figure 3: **Left:** An example of a synthesis problem given by two binary parameters (i.e., the size of the design space is four) inducing three different Markov chains. **Right:** The coloured MDP $\mathcal{M}$ with colouring κ encoding the synthesis problem.

includes four parameter assignments inducing three MCs (the assignments δ_3 and δ_4 induce the same MC). Note that transitions emanating from state 0 depend on the assignment of both parameters, while the transitions for other states are determined by a single parameter. The right part of the figure shows the resulting coloured MDP $\mathcal{M}$ with colouring κ. (The action names are chosen arbitrarily.) The colour code in the MDP illustrates the parameter dependencies given by κ. For example, action α_0 in state 0 is coloured by $h_1 = 0$ (orange) and $h_2 = 0$ (blue).

Coloured MDPs enable the (compact) encoding of various synthesis problems. In this paper, we focus on the synthesis of probabilistic programs from program sketches and the synthesis of finite-state controllers (FSCs) for various partially observable domains. We consider deterministic FSCs represented as Mealy machines that decide what action to make based on the current memory node and the current observation (Amato, Bonet, & Zilberstein, 2010b) (for a formal definition, see Appendix B). In the appendix, we provide the details of how to encode these synthesis problems either given as sketches of probabilistic programs (see Appendix A) or as synthesis problems of FSCs for (decentralised) POMDPs (see Appendix B) as coloured MDPs.

Remark 1 (Coloured MDP vs. POMDPs) *Let us clarify the relation between coloured MDPs and POMDPs, i.e., MDPs in which each state is associated with a (set of) observations. POMDP policies can select an action based on an observation history, i.e., a finite sequence of observations and actions. In contrast to MDPs, such policies can thus not use the sequence of precise states visited (and actions executed) so far. Although there are certain similarities between coloured MDPs and POMDPs, coloured MDPs can encode synthesis problems that cannot be interpreted as POMDPs. In particular, the finite design space Δ of parameter assignments does not correspond to the infinite set of observation-based policies of a POMDP. On the other hand, a POMDP and a* template *of an observation-based policy (e.g. a parameterised finite-state controller with a finite number of instantiations) can be encoded as a coloured MDP.*

2.3 Synthesis Problems

We consider a constrained single objective synthesis problem. The *constraints* are given by the formula $\Phi = \bigwedge_{i \in I} \phi_i$ where I is a finite set of indices and $\phi_i \equiv \mathbb{P}\left[\Diamond T_i\right] \bowtie_i \lambda_i$ or $\phi_i \equiv \mathbb{E}\left[\Diamond T_i\right] \bowtie_i \rho_i$ for target set $T_i \subseteq S$. Here, binary comparison operator $\bowtie_i \in \{<, \leq, >, \geq\}$, probability threshold $\lambda_i \in [0,1]$, and reward threshold $\rho_i \in \mathbb{R}$. The main synthesis objective is to minimise (or, dually, maximise) a reachability probability (or expected total reward) for a given target set T_o. We provide the formal statement for maximising reachability probabilities below; the dual problem and the variants for expected rewards are defined similarly.

Maximal reach-probability synthesis: Given a coloured MDP $\mathcal{M} = (M, \Delta, \kappa)$, constraints $\Phi = \bigwedge_{i \in I} \phi_i$, and target set T_o, find a parameter assignment $\delta^* \in \Delta$ with

$$\delta^* \in \operatorname*{arg\,max}_{\{\delta \in \Delta\}} \left\{ \mathbb{P}^{M^\delta}\left[\Diamond T\right] \mid \mathcal{M}^\delta \models \Phi \right\}.$$

Note that MC $\mathcal{M}^\delta \models \Phi$ iff $M \models \phi_i$ for all $i \in I$. Here $\mathcal{M}^\delta \models \mathbb{P}\left[\Diamond T_i\right] \bowtie_i \lambda_i$ iff $\mathbb{P}^{\mathcal{M}^\delta}\left[\Diamond T_i\right] \bowtie_i \lambda_i$ and analogously for $\mathbb{E}\left[\Diamond T_i\right] \bowtie_i \rho_i$.

We also consider a simplified synthesis problem variant, called (existentially) *feasibility synthesis*: For the constraints Φ, find a parameter assignment $\delta \in \Delta$ such that $\mathcal{M}^\delta \models \Phi$. Our framework also supports a universal variant of feasibility synthesis, which requires a finding of *all* assignments $\delta \in \Delta$ satisfying Φ.

The considered synthesis problems are a true challenge; e.g., the (existential) feasibility synthesis for families of MCs is already NP-hard (Junges, 2020). This result can be naturally lifted to the feasibility synthesis over coloured MDPs.

To simplify the exposition, we will present our approach only for reachability properties, but it can be naturally extended towards reward properties.

3. Abstraction Refinement

In order to effectively solve the aforementioned parameter synthesis problems, we introduce an abstraction-refinement (AR) scheme over the coloured MDP $\mathcal{M}$ that leverages as the oracle the model-checking of the underlying MDP M providing an abstraction of $\mathcal{M}$. This section restates our earlier versions of AR presented in (Andriushchenko et al., 2021, 2022) in the context of coloured MDPs and presents two major extensions: (a) a new splitting strategy for the refinement and (b) explicit support for multi-constrained specifications.

3.1 MDP Abstraction

We first define the notion of *policy consistency* that captures the compatibility of a policy σ in MDP M w.r.t. colouring κ.

Definition 2 (Consistent policies and relevant parameters) *Let $\mathcal{M} = (M, \Delta, \kappa)$ be a coloured MDP and σ be a policy of M. Let $K^\sigma(h) \coloneqq \{\kappa(s, \sigma(s))(h) \mid s \in S\} \backslash \{\perp\}$ be the values of parameter h used by σ. The policy σ is* consistent *if no parameter is assigned*

distinct values, i.e., $|K^\sigma(h)| \leq 1$ *for all* $h \in H$*. Furthermore, Parameter* $h \in H$ *is* relevant for σ *if* $K^\sigma(h) \neq \emptyset$*; otherwise it is irrelevant (for* σ*).*

Example 2 *Consider the coloured MDP from Fig. 3 (right) and the policy* σ_1 *with* $\sigma_1(s_0) = \alpha_0, \sigma_1(s_1) = \alpha_1$*. The policy* σ_1 *is inconsistent as* $K^{\sigma_1}(h_1) = \{0, 1\}$*, i.e.,* $|K^{\sigma_1}(h_1)| > 1$*.*

A consistent policy σ induces a (partial) parameter assignment $\overline{\delta} \in \overline{\Delta}$ with $\overline{\delta}(h) \in K^\sigma(h)$ if h is relevant and $\overline{\delta}(h) = \bot$ otherwise. Irrelevant parameter h can be assigned an arbitrary value from $D(h)$, obtaining a total assignment δ that induces the same MC: $\mathcal{M}^{\overline{\delta}} = \mathcal{M}^\delta$. Conversely, from Def. 1 it follows that each total assignment $\delta \in \Delta$ corresponds to a consistent policy σ that picks in state $s \in S$ the unique action $\sigma(s)$ for which $\kappa(s, \sigma(s)) \sim \delta$. Distinct total assignments can correspond to the same consistent policy as some parameters can be irrelevant for σ.

The following theorem shows that the set of all (possibly inconsistent) schedulers (in an uncoloured MDP) yields sound bounds for the maximum (or minimum) reachability probability over the consistent policies. This statement allows us to use the uncoloured MDP as a proper abstraction.

Theorem 1 *Let* $\mathcal{M} = (M, \Delta, \kappa)$ *be a coloured MDP and* $T \subseteq S$ *be a set of target states in the underlying MDP* M*. Then*

$$\mathbb{P}^M_{\min}[\Diamond T] \leq \min_{\delta \in \Delta} \mathbb{P}^{M^\delta}[\Diamond T] \leq \max_{\delta \in \Delta} \mathbb{P}^{M^\delta}[\Diamond T] \leq \mathbb{P}^M_{\max}[\Diamond T].$$

Proof 1 *Let* σ_δ *denote the (consistent) policy that corresponds to assignment* $\delta \in \Delta$ *and let* $\Sigma^M_\Delta := \{\sigma_\delta \mid \delta \in \Delta\}$ *be the set of all such policies. Since* $\Sigma^M_\Delta \subseteq \Sigma^M$*, it holds that*

$$\begin{aligned}\mathbb{P}^M_{\min}[\Diamond T] &= \min_{\sigma \in \Sigma^M} \mathbb{P}^{M^\sigma}[\Diamond T] \leq \min_{\sigma \in \Sigma^M_\Delta} \mathbb{P}^{M^\sigma}[\Diamond T] \\ &= \min_{\delta \in \Delta} \mathbb{P}^{M^\delta}[\Diamond T],\end{aligned}$$

and similarly for $\mathbb{P}_{\max}$*.*

Remark 2 *The correspondence between consistent policies and total assignments gives us an alternative approach for the considered synthesis problems: instead of enumerating all* $\delta \in \Delta$ *and analysing the induced* M^δ*, one can enumerate and analyse all consistent policies in the coloured MDP* $\mathcal{M}$*. However, iterating over exponentially many consistent schedulers remains ineffective in general. Another approach is to employ solving techniques for NP-complete problems, like satisfiability modulo linear real arithmetic. As we showed in our previous work (Češka et al., 2019b), this approach provides an inferior performance.*

3.2 AR for Feasibility Synthesis with a Single Constraint

We first explain AR for the setting of (existential) feasibility. Let coloured MDP $\mathcal{M} = (M, \Delta, \kappa)$ and, without loss of generality, constraint $\phi \equiv \mathbb{P}[\Diamond T] < \lambda$. Verifying the underlying MDP M yields an upper bound ub and a lower bound lb, respectively, together with their corresponding policies $\sigma_{\max}$ and $\sigma_{\min}$. As (the set of) consistent policies is subsumed by (the

set of) all policies, lb and ub are also (conservative) safe bounds for any consistent policy in $\mathcal{M}$. These bounds can now be used as follows. If $ub \leq \lambda$, then no policy, and thus no consistent policy, will exceed λ. Thus, all $\delta \in \Delta$ satisfy ϕ. Analogously, if $\lambda \leq lb$, then all $\delta \in \Delta$ violate ϕ and we can discard the whole Δ. If $lb \leq \lambda \leq ub$ and $\sigma_{\min}$ is consistent, then the abstraction is too coarse. In the latter case, one cannot draw any conclusion about the entire Δ, but $\sigma_{\min}$ represents a satisfiable $\delta \in \Delta$. Otherwise, the abstraction needs to be refined.

We thus refine if $lb \leq \lambda \leq ub$ and $\sigma_{\min}$ is inconsistent. Refinement partitions the domain $D(h)$ for a selected parameter $h \in H$ into sub-domains $D_1(h), D_2(h)$, thus decomposing Δ into sub-spaces Δ_1 and Δ_2. The goal is to obtain a more precise abstraction. We thus focus on splitting some parameter h in which $\sigma_{\min}$ is inconsistent: let $\tilde{H} = \{h \in H \mid |K^\sigma(h)| > 1\}$ be the set of such parameters. Assume $h \in \tilde{H}$ where $a, b \in K^\sigma(h)$. Observe that if we split $D(h)$ into $D_a(h)$ and $D_b(h)$ where $a \in D_a(h)$ and $b \in D_b(h)$, then policy $\sigma_{\min}$ is no longer available in the sub-abstractions. If $|\tilde{H}| > 1$, we employ the following heuristic to select h. The idea is to assign to each parameter $h \in \tilde{H}$ a *score* $v(h)$ that, roughly speaking, corresponds to the expected variance of outcomes of different actions that are associated with inconsistent assignments $K^\sigma(h)$. In other words, a high value of $v(h)$ indicates that different options of h (may) have a high impact on the reachability probability $\mathbb{P}[\Diamond T]$. We then split on the parameter with the highest score. Having obtained the refined sub-space $\Delta_i \subset \Delta$, the refined abstraction is a sub-MDP that contains only actions that are compatible with Δ_i: $M^{\Delta_i} := (S, s_0, Act, \mathcal{P}')$ for which $\mathcal{P}'(s,\alpha) = \mathcal{P}(s,\alpha)$ if $\exists \delta \in \Delta_i \colon \kappa(s,\alpha) \sim \delta$ and $\mathcal{P}'(s,\alpha) = \bot$ otherwise.

How do we compute the scores? For state s and enabled action $\alpha \in Act(s)$, let the score $v(s,\alpha) := \sum_{s' \in S} \mathcal{P}(s,\alpha,s') \cdot \mathbb{P}_{\min}(s' \models \Diamond T)$ where $\mathbb{P}_{\min}$ is obtained from model-checking the MDP abstraction M. Let $Act^\sigma(s,h) := \{\alpha \in Act(s) \mid \kappa(s,\alpha)(h) \in K^\sigma(h)\}$ be the set of enabled actions in s associated with the inconsistent assignments $K^\sigma(h)$ of parameter h. The score of parameter h in state s is:

$$v(s,h) := \max_{\alpha \in Act^\sigma(s,h)} v(s,\alpha) - \min_{\alpha \in Act^\sigma(s,h)} v(s,\alpha).$$

The overall *score* of parameter h is $v(h) := \sum_{s \in S} EV(s) \cdot v(s,h)$ where $EV(s)$ denotes the expected number of visits of state s in the induced DTMC $M^{\sigma_{\min}}$(Puterman, 1994). We thus weigh the scores for h in a state with the expected visiting time of s. These expected visiting times can be efficiently computed for all states by solving a single linear equation system (Mertens, Katoen, Quatmann, & Winkler, 2024).

3.3 AR for Feasibility Synthesis with Multiple Constraints

Feasibility synthesis for multiple constraints $\Phi = \bigwedge_{i \in I} \phi_i$ is done by adapting the usage of the abstraction as well as the splitting strategy during refinement. For each ϕ_i, we apply the MDP abstraction and model-check the resulting MDP M. If all policies in M violate some ϕ_i, we can conclude that all $\delta \in \Delta$ violate Φ. In the absence of any such ϕ_i, we simplify Φ: if all policies in M satisfy ϕ_i, we remove ϕ_i from Φ in the consequent synthesis process. For the remaining ϕ_i, we collect $\delta_i \in \Delta$ that satisfy ϕ_i and analyse their *incompatibility*. We can quantify the *level of incompatibility* by analysing the differences between the assignments. For each parameter $h \in H$, let $L(h) = |\{\delta_i(h) \mid i \in I\}|$ denote the level of incompatibility.

When dealing with multiple constraints, we use the level of incompatibility $L(h)$ instead of the overall score $v(h)$. For the refinement, a high $L(h)$ indicates that different constraints rely on different options of h, and therefore, restricting the options of h might have a high impact on the feasibility of these constraints. Splitting according to the maximal level of incompatibility ensures that we will find different policies (compared to the currently found policies) for some of the constraints in the sub-domains and their respective sub-spaces. Assume that we chose $h \in H$ with $L(h) = 2$ where $\delta_1(h) = \{a\}$ and $\delta_2(h) = \{b\}$. Observe that by splitting $D(h)$ into $D_a(h)$ and $D_b(h)$ where $a \in D_a(h)$ and $b \in D_b(h)$, we obtain sub-space Δ_a and Δ_b such that $\delta_1 \notin \Delta_b$ and $\delta_2 \notin \Delta_a$.

3.4 AR for Maximal Reach-probability Synthesis

The maximal reachability probability synthesis wrt. a target T can be formulated as a chain of feasibility synthesis queries where the constraint $\mathbb{P}[\Diamond T] > \lambda$ is iteratively strengthened. Note that each time the constraint is strengthened, we do not need to check parts of the design space that were pruned before, as we already proved they are unfeasible for weaker constraints. Initially, we put $\lambda = 0$. If a solution $\delta \in \Delta$ is found such that $\mathbb{P}^{M^\delta}[\Diamond T] > 0$, we store δ as a new running optimal solution and update the threshold $\lambda = \mathbb{P}^{M^\delta}[\Diamond T]$. This process continues for the new threshold λ, which is updated every time a new running optimal solution is found. The synthesis terminates if the entire design space Δ is explored and pruned. When the synthesis terminates, the running optimal solution is the actual optimal solution and is the output of the synthesis.

4. Counterexample-Guided Synthesis

The AR algorithm described in the previous section allows us to explore large design spaces. However, the MDP abstractions built during the exploration are sometimes significantly larger (in terms of the reachable underlying state space) and numerically harder to solve than any of the candidate solutions M^δ, rendering the analysis of the MDP too expensive. This motivates an orthogonal approach using a cheaper oracle in the form of a counter-example generator over candidate solutions. The key idea is to generalise a candidate solution $\delta \in \Delta$ that refutes the specification Φ to a set of structurally similar candidates $G(\delta) \subseteq \Delta$ that all refute Φ. The main advantage is that while exploring the design space, all candidate solutions in $G(\delta)$ can be safely ignored once one finds out that δ refutes Φ. In contrast to the top-bottom AR approach – by analysing sets of candidates (modelled as MDPs), this set is gradually refined – the generalisation G provides a bottom-up approach – by analysing single candidates (Markov chains), sets of candidates are ruled out. This section summarises the key concepts from our earlier work (Češka et al., 2021; Andriushchenko et al., 2021) and lifts them into coloured MDPs.

4.1 Counterexamples for Probabilistic Systems

Let ϕ be a single constraint. Intuitively, a *counterexample* for MC $D \not\models \phi$ is a sub-MC D' of D that refutes ϕ and it is minimal w.r.t. refuting ϕ if any extension of D' refutes ϕ (Wimmer et al., 2014).

Definition 3 (Counterexample) *The* sub-MC *of MC $D = (S, s_0, \mathcal{P})$ induced by the set $C \subseteq S$ of states is the MC $D{\downarrow}C = (S \cup \{s_\bot\}, s_0, \mathcal{P}')$ where $s_\bot \notin S$ and*

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

The sub-MC $D{\downarrow}C$ is a counterexample *(CE) for (safety) constraint $\phi \equiv \mathbb{P}[\Diamond T] < \lambda$ if $D{\downarrow}C \models \mathbb{P}[\Diamond T] \geq \lambda$. The sub-MC $D{\downarrow}C$ is a CE for (liveness) constraint $\mathbb{P}[\Diamond T] > \lambda$ if $D{\downarrow}C \models \mathbb{P}[\Diamond T_\bot] \leq \lambda$ where $T_\bot = T \cup \{s_\bot\}$.*

The notion of CE can be lifted to total expected reward objectives in a natural manner, see (Quatmann et al., 2015). A critical sub-MC serves as important diagnostic information about the source of the violation in probabilistic systems (Ábrahám et al., 2014). The CE is used to construct the generalisation function G. Let $\delta \in \Delta$ be a violating parameter assignment w.r.t. the coloured MDP $\mathcal{M}$ and constraint ϕ, i.e., $\mathcal{M}^\delta \not\models \phi$. To compute the generalisation $G(\delta)$ that includes other assignments violating ϕ, we identify the set $H_C^\delta \subseteq H$ of *relevant parameters* (called *conflict*) for the CE $\mathcal{M}^\delta{\downarrow}C$. Let $H(s,\alpha) \coloneqq \{h \in H \mid \kappa(s,\alpha)(h) \neq \bot\}$ denote the set of parameters defined for action α in state s and $A(s,\delta) = \alpha \iff \kappa(s,\alpha) \sim \delta$, i.e., $A(s,\delta)$ denotes the enabled action in s in the MC M^δ.

Definition 4 (Conflict and CE generalisation) *Let coloured MDP $\mathcal{M} = (M, \Delta, \kappa)$, violating assignment $\delta \in \Delta$ and CE $\mathcal{M}^\delta{\downarrow}C$. The* conflict *is $H_C^\delta \coloneqq \bigcup_{s \in C} H(s, A(s,\delta))$. The* generalisation of δ w.r.t. the CE $\mathcal{M}^\delta{\downarrow}C$ *is defined as $G_C(\delta) \coloneqq \{\delta' \in \Delta \mid \forall h \in H_C^\delta : \delta(h) = \delta'(h)\}$.*

Each $\delta' \in G_C(\delta)$ can be safely removed from Δ, as it is guaranteed that δ' violates ϕ. The next section describes how $G_C(\delta)$ is used in the synthesis algorithm.

The size $|H_C^\delta|$ of a conflict is crucial. Small conflicts lead to generalising δ to a larger set of assignments. It is thus important that the CEs contain a minimal number of parameterised transitions. Note that the size of a CE in terms of its number of states is not relevant.

The CE generalisation can be straightforwardly lifted to more involved specifications. For multiple constraints $\Phi = \bigwedge_{i \in I} \phi_i$, we can compute and generalise the CE for each ϕ_i that is violated by the candidate M^δ. Having multiple CEs for a single M^δ is indeed beneficial for pruning but requires more computation; for details, see (Češka et al., 2021). The (constrained) optimisation is handled similarly as in AR, i.e., as a chain of feasibility queries.

4.2 Counterexamples Modulo Bounds

The conflict size can be further reduced by *CE modulo bounds* (Andriushchenko et al., 2021). Assume $\phi \equiv \mathbb{P}[\Diamond T] < \lambda$. For a given sub-space Δ, the abstraction provides upper bounds ub_ϕ^Δ such that $\forall \delta \in \Delta, s \in S : ub_\phi^\Delta(s) \geq \mathbb{P}^{M^\delta}[s \models \Diamond T]$. Intuitively, using ub_ϕ^Δ in the CE construction can lead to smaller CEs since they provide less conservative bounds on reachability probabilities for states $s \in S \backslash C$.

Definition 5 (Counterexample modulo bounds) *Assume MC $D = (S, s_0, \mathcal{P})$ and let ub_ϕ^Δ be the upper bounds as above. The* sub-MC *of D induced by $C \subseteq S$ and ub_ϕ^Δ is*

$D{\downarrow}C(ub_\phi^\Delta) = (S \cup \{s_\bot, s_\top\}, s_0, \mathcal{P}')$ *where* $s_\top, s_\bot \notin S$ *and*

$$\mathcal{P}'(s) = \begin{cases} \mathcal{P}(s) & \text{if } s \in C, \\ [s_\top \mapsto ub_\phi^\Delta(s), s_\bot \mapsto (1{-}ub_\phi^\Delta(s))] & \text{if } s \in S\backslash C, \\ [s \mapsto 1] & \text{otherwise .} \end{cases}$$

The sub-MC $D{\downarrow}C(ub_\phi^\Delta)$ *is called a* counterexample *(CE) for* ϕ*, if* $D{\downarrow}C(ub_\phi^\Delta) \not\models \mathbb{P}[\Diamond T] < \lambda$.

CE generalisation using the CE modulo bounds is safe, i.e., it includes only the violating assignments. Formally:

Theorem 2 *Let* δ *be an assignment such that* $M^\delta \not\models \phi$*,* $M^\delta{\downarrow}C(ub_\phi^\Delta)$ *be a CE modulo bounds, and* $G_C(\delta)$ *be the corresponding generalisation. For each* $\delta' \in G_C(\delta)$*,* $M^{\delta'} \not\models \phi$.

Proof 2 *Assume arbitrary* $\delta' \in G_C(\delta)$*. We argue that* $M^{\delta'}{\downarrow}C(ub_\phi^\Delta) = M^\delta{\downarrow}C(ub_\phi^\Delta)$*: if* $s \in C$*, then* $\mathcal{P}_\delta(s) = \mathcal{P}_{\delta'}(s)$ *follows from the definition of* $G_C(\delta)$*, otherwise the equality of both transition probability distributions in* s *is trivial. Thus,* $M^{\delta'}{\downarrow}C(ub_\phi^\Delta) \not\models \phi \iff M^\delta{\downarrow}C(ub_\phi^\Delta) \not\models \phi$*. Since* ub_ϕ^Δ *are the upper bounds for* δ *as well as* δ'*, then* $M^{\delta'}{\downarrow}C(ub_\phi^\Delta)$ *must be a valid counterexample (modulo bounds) for* $M^{\delta'}$*, i.e.* $M^{\delta'} \not\models \phi$.

4.3 Greedy Algorithm for CE Generation

Finding CEs of minimal size is NP-complete (Funke, Jantsch, & Baier, 2020). Existing CE procedures (Ábrahám et al., 2014; Dehnert et al., 2014; Wimmer et al., 2015) do not provide good trade-offs between the conflict size and their construction time. We, therefore, employ a greedy algorithm (Andriushchenko et al., 2021) and adapt it to coloured MDPs. Assume MC $\mathcal{M}^\delta$ refutes constraint ϕ. A greedy approach to construct the conflict set H_C^δ computes a sequence of gradually increasing candidate conflict sets H_i. We conservatively start with $H_0 = \emptyset$. At the i-th iteration, we consider $C(H_i) \coloneqq \{s \in S \mid H(s, Act(s,\delta)) \subseteq H_i\}$, the set of states for which the colouring of the selected action $Act(\cdot,\delta)$ includes only parameters from H_i. If $C(H_i)$ induces a CE $\mathcal{M}^\delta{\downarrow}C(H_i)$ for ϕ, then H_i is the corresponding conflict set. Otherwise, we enlarge H_i as follows: $H_{i+1} \coloneqq H_i \cup H(s_i, Act(s_i,\delta))$ for some arbitrary state $s_i \in S\backslash C(H_i)$ that is reachable in $\mathcal{M}^\delta{\downarrow}C(H_i)$. This iterative approach is continued until the set $C(H_i)$ induces a CE for ϕ, for some i. As $\mathcal{M}^\delta \not\models \phi$, this procedure terminates.

5. Synthesis Algorithm

This section summarises the overall architecture and workflow of the synthesis algorithm. Fig. 2 illustrates the main components and their interaction. The algorithm builds on two synthesis engines repeatedly invoking the respective oracles: AR (described Sect. 3) and CEG (described in Sect. 4). The workflow consists of three main parts: manager, explorer and evaluator. The manager encodes the given synthesis problem into a coloured MDP $\mathcal{M} = (M, \Delta, \kappa)$ (see Def. 1). It also controls the synthesis process, collects and interprets the results from the engines, and produces the final results.

The explorer maintains the parts of the design space Δ that have to be explored. These parts are independently stored using a stack of sub-spaces and an SMT formula φ encoding the unexplored parameter assignments.

In every AR iteration, a sub-space Δ_i is processed by the evaluator that constructs the corresponding MDP abstraction M (see Sect. 3.1) and performs a MDP model-checking for each constraint $\phi_i \in \Phi$. The sub-space Δ_i is discarded if it violates any ϕ_i, or is further analysed w.r.t. consistency and compatibility (see Sect. 3.3). If the analysis is inconclusive (i.e., the abstraction is too coarse), Δ_i is refined into a set of sub-spaces that are pushed on the stack.

In every CEG iteration, an SMT solver (we use Z3 (de Moura & Bjørner, 2008) or CVC5 (Barbosa et al., 2022)) is asked to produce an assignment δ satisfying φ, i.e., an unexplored assignment. The evaluator then constructs the corresponding MC M^δ and checks whether it satisfies each $\phi_i \in \Phi$. Otherwise, for each violated ϕ_i, the evaluator constructs the CE using the bounds for δ (see Def. 5) and the corresponding generalisations $\mathrm{G}(\delta)$ (see Def. 4). The resulting set of generalisations is encoded as a set of clauses describing new constraints on the unexplored assignments and passed to the explorer that updates the SMT formula φ.

PAYNT further supports the so-called *hybrid* engine (Andriushchenko et al., 2021) that effectively combines AR and CEG. This engine leverages CE modulo bounds (see Section 4B), where the bounds on the probabilities/rewards computed in AR for particular sub-spaces are used in CEG. This enables obtaining better, i.e., larger generalisations of CEs and thus prunes design sub-spaces more aggressively. The manager distributes the workload and switches between the AR and CEG engines. It facilitates refining during the AR phase and using the obtained refined bounds during CEG. Additionally, it exploits an efficiency measure of the two methods and dynamically allocates more time to the method with superior performance.

Algorithm 1 summarises the key steps of the hybrid synthesis algorithm for the maximal reach-probability problem (see Section 2.3). We start ($\mathtt{l}$.1) by creating the sub-space stack (initially containing only Δ) and construct the formula φ describing each unexplored parameter assignment from Δ. In each iteration of the main loop, we explore exactly one sub-space Δ_i. We first run one iteration of AR ($\mathtt{l}$.4), which may update the current optimum δ^*. This call also yields probability reachability bounds for Δ_i. If this one iteration proves that the sub-space does not contain a better solution than (updated) δ^*, we continue with the next sub-space. Otherwise, we set the time limit for CEG ($\mathtt{l}$.6) based on the previous performance of AR and CEG. When exploring Δ_i, CEG uses $\mathtt{bounds}_i$ to construct counterexamples modulo bounds. CEG terminates either by exploring all assignments from Δ_i or by reaching the allocated time limit. In the latter case ($\mathtt{l}$.9), we split Δ_i into sub-families to be analysed in subsequent iterations.

To balance AR vs. CEG reasoning, we track the number $\eta_{\mathtt{AR}}$ and $\eta_{\mathtt{CEG}}$ of assignments pruned per time unit by AR and CEG, respectively. Let $\tau_{\mathtt{AR}}$ and $\tau_{\mathtt{CEG}}$ denote the current total run-times of AR and CEG; then, on $\mathtt{l}$.6 we allocate $\tau_{\mathtt{AR}} \cdot \frac{\eta_{\mathtt{CEG}}}{\eta_{\mathtt{AR}}} - \tau_{\mathtt{CEG}}$ time units to CEG to explore Δ_i. The idea is to let AR and CEG run for equal portions of time when both are equally successful and to allocate more time to CEG when it outpaces AR in terms of the number of pruned assignments, and vice versa.

Theorem 3 *Algorithm 1 terminates and returns assignment $\delta^* \in \Delta$ satisfying the given constraints and attaining the maximal reach-probability as defined in Section 2.3.*

Proof 3 *Assume an iteration of the algorithm in which sub-space Δ_i is investigated and assume it contains an optimal assignment δ^* satisfying the constraints. Assume that δ^* has*

Algorithm 1: Hybrid synthesis

Global : A coloured MDP $\mathcal{M} = (M, \Delta, \kappa)$, constraints $\Phi = \bigwedge_{i \in I} \phi_i$, and target set T_o

Output : $\delta^* \in \Delta$ as defined in Section 2.3

```
1  stack.new(Δ), φ ← encode(Δ), δ* ← ∅;
2  while not stack.empty() do
3      Δ_i ← stack.pop();
4      bounds_i, δ* ← AR.run(Δ_i, δ*);
5      if AR.done() then continue;
6      CEG.setTimeLimit();
7      φ, δ* ← CEG.run(φ, Δ_i, bounds_i, δ*);
8      if CEG.done() then continue;
9      stack.push(AR.split(Δ_i))
10 end while
11 return δ*
```

value v^* and that $v \leq v^*$ is the value of the running optimum. To show correctness, we argue that Δ_i cannot be pruned by AR or CEG without first identifying δ^*.

Generalising Theorem 1 to the given set of reach-probability constraints (see Sections 3.3 and 3.4), AR call on 1.4 can prune Δ_i by one of two ways: i) all $\delta \in \Delta_i$ violate some $\phi \in \Phi$, or ii) it holds that $\mathbb{P}^{M^{\Delta_i}}_{\max}[\Diamond T_o] < v$, i.e. $v^* = \mathbb{P}^{M^{\delta^*}}[\Diamond T_o] < v$ – both cases contradict the fact that δ^* is the optimal assignment satisfying the constraints.

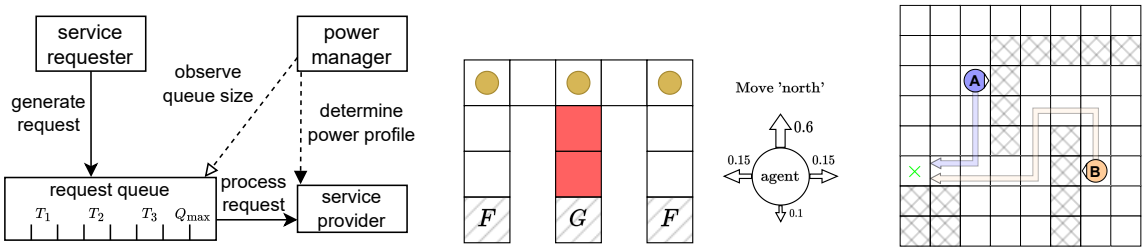
Now assume the CEG phase that currently investigates an assignment $\delta \in \Delta_i$, $\delta \neq \delta^*$. If δ violates some $\phi \in \Phi$, a conflict $G_C(\delta)$ is constructed and from Theorem 2 we know that if $\delta^* \in G_C(\delta)$, then $M^{\delta^*} \not\models \phi$, which is a contradiction. Otherwise, if all constraints are satisfied, a counterexample is constructed wrt. $\mathbb{P}^{M^{\delta}}[\Diamond T_o] \leq v$, and the corresponding conflict again cannot contain δ^* unless $v = v^*$, i.e. an assignment with the same value has already been found.

To show termination, observe that the CEG reasoning is always bound by a time limit and that without lines 6-8 the algorithm is a standard AR loop in which every sub-space Δ_i is either pruned or split into strictly smaller sub-spaces. Since the design space Δ is finite, the smallest possible sub-space Δ_i contains a single parameter assignment, for which AR reasoning on 1.4 is guaranteed to be conclusive.

6. Experimental Evaluation

This section presents an experimental evaluation of our approach. It mainly focuses on three questions:

Q1 *Is our approach able to solve interesting problems from different planning and decision-making domains?* To answer this, we consider three case studies: i) synthesising a power manager from a rough sketch, ii) constrained planning in a maze POMDP and iii) solving a meeting-grid Dec-POMDP. We discuss and interpret the resulting policies.



(a) (b) (c)

Figure 4: (a) Overview of the DPM system. (b) Constrained Maze problem and the considered slipping factors for this problem. (c) The considered instance of the 8x8 meeting grid Dec-POMDP problem with two agents and their starting positions.

Q2 *Is our approach competitive with respect to dedicated methods for the aforementioned planning and decision-making problems?* To answer this, we compare the performance of our approach with the state-of-the-art methods to synthesise FSCs for i) constrained POMDPs and ii) Dec-POMDPs.

Q3 *How do the synthesis engines (i.e., AR, CEG, and their symbiosis) perform and how do the proposed enhancements (namely the splitting strategy for the coloured MDPs) improve the performance with respect to the previous version of PAYNT (Andriushchenko et al., 2021), a state-of-the-art tool to synthesise finite-state probabilistic programs.*

Hardware: All experiments are run on a single core of a machine equipped with an Intel i5-12600KF @4.9GHz CPU with 64GB RAM.

Availability: The version of PAYNT used for the evaluation, as well as all benchmarks, are publicly available (Andriushchenko, Češka, Junges, Katoen, & Macák, 2024).

Q1: Case Studies from Different Planning and Decision-making Domains

Synthesising a power manager from a rough sketch

Recall the Dynamic Power Management (DPM) problem described in Section 1.2. We first consider a simple variant with a single queue, see Fig. 4a, and a service provider with three power modes: *sleep*, *idle* and *active* with increasing power consumption that depends on the queue size, random switching latency, and random error of request processing (only the active mode allows for request processing). In each step, the system terminates with a probability of 0.001 which amounts to 648.5 requests to be processed on average. The objective—the specification Φ—is to minimise the power consumption during the operation time while ensuring that the expected number of lost requests is at most one. The specification is supplemented by a sketch describing the set of candidate solutions. We use a sketch for a finite-state probabilistic program represented in the PRISM guarded-command language (Kwiatkowska et al., 2011). (Such sketches are not new; they were proposed for programs with uncertainty (Češka et al., 2021).) *Holes* in the program sketch represent the program's unknown parts. Each hole is associated with a finite domain representing the possible options (i.e., program snippets) that can fill in the hole. The goal of the PM

synthesis is to find a hole assignment for its sketch such that the resulting completed program satisfies specification Φ.

The PM cannot fully observe the request queue. Instead, it can only distinguish four occupancy levels determined by the threshold parameters $T_1 \in \{0, 0.1, 0.2, 0.3, 0.4\}$, $T_3 \in \{0.6, 0.7, 0.8, 0.9\}$, and a fixed threshold T_2=0.5[4]. As the power consumption depends on the queue size, we aim to obtain the best power mode for each occupancy level. This is represented by parameter P_i (for occupancy level i) whose domain is $\{\text{sleep}, \text{idle}, \text{active}\}$. The maximal capacity QMAX ranges over $\{1, 2, \ldots, 10\}$. As power consumption depends on the queue size, choosing QMAX=10 is not trivially optimal.

The design space consists of $5 \times 4 \times 3^4 \times 10 = 16,200$ candidate programs, i.e., policies. Every candidate program is a probabilistic program, which corresponds to an MC (of about 230 states on average). Within about 3 minutes, our approach finds the following optimal solution: QMAX = 6, $T_1 = 0.1$, $T_3 = 0.9$, $P_1 = \text{idle}$, P_2=P_3=P_4=active. This policy induces a power consumption of 95,531 Watt. The expected number of lost requests is 0.617 during operation. The trivial "safe" policy using a maximal queue size (QMAX=10) and setting the power mode to be always active increases the power consumption by about 7%.

Let us now consider the more complicated variant of the DPM problem that distinguishes low and high priority requests (and puts these in two separate queues) and a longer expected runtime. The design space of this variant contains over 43 million candidate solutions. Compared to the single-queue setting, the average size of the underlying Markov chain is more than one order of magnitude larger.

Our approach needs about 6.5 hours to explore all candidates, while the trivial (one-by-one) checking of all candidate solutions would take over a month. The optimal solution improves the power consumption of the system by 30% compared to the trivial "always active" solution. Allowing a bit of tolerance in the optimality of the policy-to-be-synthesised speeds up the synthesis significantly. To that end, our approach supports ϵ-optimal policy synthesis. This returns a sub–optimal policy that is optimal with respect to the specification Φ up to a factor ϵ. Within 7 minutes, our approach finds an 0.002-optimal policy for the DPM variant with two request queues i.e. the obtained power consumption is within 0.2% of the optimal solution. Being just a bit liberal with respect to the quality of the policy thus can speed up the synthesis process enormously.

Constrained planning in a maze POMDP

The previous case study demonstrated that our approach can effectively handle constrained optimisation in coloured MDPs describing sketches of probabilistic programs. We now show that our approach can also handle constrained optimisation in POMDPs. Solving constrained POMDPs (CPOMDPs) is a notoriously difficult problem that naturally arises in many applications (Isom, Meyn, & Braatz, 2008; Wray & Czuprynski, 2022).

We consider the synthesis of an optimal FSC for a CPOMDP variant of the classical maze problem (Hauskrecht, 1997); see Fig. 4b (left). The maze has goal state G and two trap states (denoted by F, for forbidden) where the agent gets stuck. The agent only observes the walls directly next to its current position. Initially, the agent is uniformly placed onto one of the cells (neither F nor G). In every step, the agent selects one out of four directions

4. This ensures that $T_1 < T_2 < T_3$.

(north, east, west, south). Although the agent cannot decide to stay in the current cell, it may remain where it is, as the grid is slippery, and a selected move may fail by a slipping factor, see Fig. 4b (right).

For fixed k, the goal ϕ_0 is to find the k-FSC (finite state deterministic controller with k memory nodes; see the formal definition in App. B) that maximises the reachability probability to G. Additionally, we impose two antagonistic constraints: $\phi_1 :=$ the expected number of visits of the yellow cells with 'coins' (denoted E_y) is at least 10, and $\phi_2 :=$ the expected number of visits of the "danger" red cells (denoted E_r) is below 7. Although the CPOMDP is relatively small, the synthesis problem is challenging due to some trade-offs. The constraint ϕ_1 is incompatible with the objective to maximise reaching G. In addition, to minimise reaching F, the agent intends to stay close to the middle column of the maze. However, this conflicts with constraint ϕ_2.

PAYNT proves within 1 sec that neither a memoryless controller (1-FSC) nor a 2-FSC can achieve the objective under the given constraints. PAYNT finds the best 3-FSC in 42s and requires two and a half minutes to determine that there is no better 3-FSC by exploring the design space consisting of all $4.1 \cdot 10^{11}$ candidates. The obtained controller reaches the target G with probability 0.943 while $E_y = 10.84$ and $E_r = 6$.

The design space for k=4 consists of $1.8{\cdot}10^{16}$ candidates. In about 9 minutes, PAYNT finds a 4-FSC that improves the reachability probability to 0.946 and satisfies both constraints with $E_y = 10.002$ and $E_r = 6.02$. In the given 30-minute deadline, PAYNT explores 8% of all 4-FSCs, while not improving. The 4-FSC aims to reach one of the top (yellow) corners as fast as possible, and once a corner is reached – corners have unique observations – the controller tries to go back and forth in the top cells. The memory is needed to determine when the agent can switch its focus to reaching G. The controller uses the slipping factor to its advantage by going back and forth as there is around a 20% chance of reaching the middle cell. This gives the FSC enough time to satisfy the constraint on visiting the yellow states before reaching G.

Solving a meeting-grid Dec-POMDP

Recall the meeting-grid Dec-POMDP problem described in Section 1.2 where the goal is to synthesise a pair of strategies, one for each of the two agents. The agents must meet at some arbitrary cell in the grid[5] while each agent has only limited information about its position and there is no communication between the agents. This is a classical but challenging, decentralised planning problem already for 8x8 grids with obstacles (Dibangoye et al., 2012). We consider a variant that is illustrated in Fig. 4c[6]. There are 48 possible positions for each agent, resulting in $48^2 = 2{,}304$ states – this is already a very large Dec-POMDP compared to the classical Dec-POMDP benchmarks (see the comparison with state-of-the-art methods presented in Q3, namely Table 2).

We consider each agent's policy to be represented as a finite-state deterministic controller with k memory nodes (k-FSCs) (Hansen, 1998), a common idea for POMDPs. The synthesis problem is then to derive two k-FSCs, one for each agent, that jointly establish the specification Φ: to minimise the expected number of steps until the agents meet.

5. Both agents move within one discrete step, and therefore, they can jump over each other.
6. The boundaries of the grid are treated as obstacles.

We first consider a scenario in which both agents observe all obstacles within distance one (north, south, east and west). Within one second, our approach yields that it is impossible to steer the agents such that they almost surely meet using a pair of memoryless controllers, i.e., FSCs with $k = 1$. For k=2, there are 10^{18} candidates[7], and our approach (exploring all pairs of 2-FSCs) finds the optimal solution in 15 seconds.

The problem becomes more challenging for a slippery grid in which, with probability 0.1, an agent fails to move if it wants to. Our approach now requires 10 minutes to find an optimal pair of 2-FSCs. As for the perfect non-slippery grid, the agents find the shortest path to the other agent's starting location while meeting somewhere on the way.

Finally, we consider an even more challenging setting in which the grid is slippery and the observability is further limited: agent A only sees the obstacle directly to the east while agent B only sees the obstacle directly to the west. Again, the impossibility of a pair of memoryless FSCs is found within a second. We find the optimal controller for k=2 within 17 minutes. In contrast to the case above, the agents are now not able to find the shortest path to the other agent's starting location. Instead, they "agree" on the best meeting location that both agents can reach. The optimal 2-FSC controller is shown in Fig. 4c and requires an average of 9.77 steps. (The value of the optimal policy in the setting with better observability is 6.76). Surprisingly, agent A does not take the fastest route to the "agreed" location, but it alternates between staying at the current position and moving. This is caused by the fact that agent A would get there much quicker than agent B. However, in this grid, it is beneficial to meet *on the way to* the agreed location.

Q2: Comparison with State-of-the-art Controller-based Methods

In this section, we provide a comparison with other state-of-the-art controller-based approaches for the domains of i) constrained POMDP and ii) Dec-POMDPs. Our methods are focused on the synthesis of FSCs for these problems, and therefore, the comparisons are aimed at determining the performance capabilities of our approach compared to other approaches that also produce FSCs. We note that our FSCs are deterministic Mealy machines where the next action is chosen based on the memory node and the current observation. For completeness of the corresponding sections, we also mention other approaches, most notably belief-based methods (Poupart et al., 2015; Helmeczi et al., 2023; Dibangoye et al., 2016). These methods can provide good scaling but lack in terms of explainability of their output compared to methods that produce FSCs. The comparison with belief-based methods is beyond the scope of this paper.

Synthesising FSCs for constrained POMDPs

The constrained maze POMDP case study showcased our approach is able to tackle constrained POMDPs (CPOMDPs). As far as we are aware, our approach is the first one to handle indefinite-horizon (*undiscounted*) specifications in CPOMDPs. To provide a comparison to state-of-the-art, we consider specifications *with discounting*. Infinite-horizon CPOMDPs are a challenging planning problem, and current state-of-the-art algorithms struggle to scale mainly because of optimisation complexities that are related to ensuring that the constraints are satisfied. Our approach tries to find optimal FSCs strictly satisfying

7. The size of the underlying MCs ranges from 4 to 9216 states.

					CNLP		PGA		PAYNT	
Benchmark	$\|S\|$	$\|A\|$	$\|\Omega\|$	β	ADR/ADC	time	ADR/ADC	time	ADR/ADC	time
milos-97	20	6	8	10	23.2/10.0	741s	16.9/10.0	564s	**31.6**/10.0	**7s**
tiger-grid	36	5	17	18	−0.25/19.4	5685s	−0.76/19.9	735s	**0.0**/0.0	**1s**
query-s3	27	3	3	53	261.2/14.6	3s	344.1/8.7	166s	**509.5**/15.7	1453s
query-s4	81	4	3	60	-	TO	342.4/8.8	336s	**456.9**/18.4	791s

Table 1: Performance comparison of PAYNT with constrained linear programming (CNLP) and projected gradient ascent (PGA) on CPOMDPs. The table indicates the number $|S|$ of states, the number $|A|$ of actions, the number $|\Omega|$ of observations and the cost threshold β. For each approach, the table lists the average discounted reward (ADR) and average discounted cost (ADC) for the best found policy and the time to find this policy. Numbers for CNLP and PGA were taken from (Wray & Czuprynski, 2022).

the given constraints. The notable controller-based approaches we consider in our experimental comparison are constrained non-linear programming (CNLP) (Amato, Bernstein, & Zilberstein, 2010a) and projected gradient ascent (PGA) (Wray & Czuprynski, 2022) which produce stochastic FSCs. In Section 7, we discuss alternative approaches for constrained POMDPs.

Benchmarks and setup. We evaluate our approach on traditional CPOMDP benchmarks. We use single-objective POMDPs that are augmented with costs for low-reward actions taken from (Poupart et al., 2015). The timeout for each experiment was 2 hours. We report the values for the policy that achieved the best average discounted reward (ADR) within this timeout. Our baselines are CNLP and PGA, where the values were taken from (Wray & Czuprynski, 2022).

Results. The results are presented in Tab. 1. Our approach produces controllers that achieve better rewards while respecting the cost constraint. In fact, our approach produces better controllers compared to CNLP and PGA in under one second for all considered models and gradually finds even better solutions. Moreover, our approach guarantees that the constraint will be satisfied meanwhile, both CNLP and PGA fail to do so on the *tiger-grid* benchmark (note that ADC $> \beta$ in the table). Paper (Wray & Czuprynski, 2022) considers controller sizes of 5 and 10 nodes. From the results in Tab. 1, all CNLP values are for controllers with 5 nodes and for PGA it is controllers with 10 nodes for all but the *query-s4* model. For PAYNT the sizes are as follows: 1 node for *tiger-grid*, 2 nodes for *milos-97* and *query-s4*, and 3 nodes for *query-s3*. From this, we observe that PAYNT is capable of producing better controllers that are smaller or at least comparable in size.

Synthesising FSCs for Dec-POMDPs.

The meeting-grid case study showcased that our framework can be used to solve Dec-POMDPs. Solving Dec-POMDPs is a notoriously difficult problem. Many state-of-the-art approaches support infinite-horizon specifications. However, we believe our approach is the first one to solve and produce FSCs for indefinite-horizon (i.e. undiscounted) expected reward

specifications in Dec-POMDPs. The most notable algorithm that produces FSC policies for discounted infinite-horizon specifications is InfJESP (You, Thomas, Colas, & Buffet, 2021). In Section 7, we discuss alternative approaches for Dec-POMDPs.

Benchmarks and setup. To showcase that our approach is competitive for Dec-POMDP solving, we compare PAYNT with InfJESP on traditional Dec-POMDP benchmarks taken from http://masplan.org/problem_domains (Recycling, Grid3x3, DecTiger, BoxPushing) as well as on the already mentioned meeting-grid Dec-POMDP. In all models, the goal is to maximise the expected discounted reward. InfJESP searches for joint equilibria and uses random initialisation. Therefore, its performance can differ dramatically between runs. We run three InfJESP attempts with 100 restarts each and report the numbers for the best attempt. In PAYNT, we iteratively explore bigger FSC spaces, starting with 1×1 FSCs and continuing to 2×2 and so on. We keep the value of the best-found FSC from each previous computation to serve as bounds for FSC pruning.

Results. The comparison with InfJESP is shown in Tab. 2. The results show that PAYNT is competitive for solving infinite-horizon Dec-POMDPs while providing compact, explainable and easy-to-verify solutions. On the models *Recycling* and *Box-pushing*, our approach finds better FSCs compared to InfJESP while needing significantly less time (6-10× less). The *DecTiger* benchmark is a problem where our approach struggles due to the complexity of the abstraction and because a good FSC requires a lot of memory nodes. The complexity of the abstraction stems from the facts that 1) PAYNT does not work with probabilistic observations and therefore needs to unfold them into the state space[8] 2) in *DecTiger* nearly every action contains transitions to all of the (unfolded) states meaning the abstraction contains a lot of transitions. The abstraction gets even more complex with the addition of memory, as such, this is a known limitation of our synthesis loop that we encounter on domains requiring a lot of memory. On the other hand, our approach is able to quickly find compact FSCs that provide values comparable or even better compared to the state-of-the-art for most of the traditional benchmarks as shown, in particular, for the *Box-pushing* problem, our approach finds compact 2-state FSCs that improve the previously best-known value for this model. For the *Grid3x3* model, PAYNT finds a pair of memoryless FSCs[9] that are enough to match the best-known value on the model.

Q3: Comparison of the Synthesis Engines and Improvements to PAYNT

Finally, we showcase the different pros and cons of the individual synthesis engines in PAYNT presented in Sections 3, 4, and 5. We also show the improvements in terms of synthesis performance compared to our previous version of PAYNT (Andriushchenko et al., 2021).

Benchmarks and Setup

We evaluate the synthesis engines of PAYNT on various synthesis problems from the literature, extensions thereof, and a few new problems (e.g., the meeting-grid Dec-POMDP (Dibangoye et al., 2012) synthesis problem). Our benchmark suite includes some of the models (indicated by †) considered in the experimental evaluation of the previous version of PAYNT (An-

8. From our benchmark set only *DecTiger* uses probabilistic observations.
9. Recall we represent FSCs as Mealy machines.

						InfJESP			PAYNT		
Benchmark	$\lvert\mathcal{I}\rvert$	$\lvert S\rvert$	$\lvert A^i\rvert$	$\lvert\Omega^i\rvert$	γ	value	FSC size	time	value	FSC size	time
Recycling	2	4	3	2	0.9	31.5	2×2	6s	**31.9**	2×2	<1s
Grid3x3	2	81	5	9	0.9	5.82	8×8	270s	5.82	1×1	**21s**
DecTiger	2	2	3	2	0.9	**8.22**	43×48	4160s	-2.28	3×3	1900s
Box-pushing	2	100	4	5	0.9	219.2	85×89	654s	**224.7**	2×2	46s
Meeting-grid	2	4096	4	2	0.99	**−6**	11×13	12488s	−7.52	3×3	450s

Table 2: Performance comparison of PAYNT with InfJESP on infinite-horizon Dec-POMDPs. The table lists the number $\lvert\mathcal{I}\rvert$ of agents, the number $\lvert S\rvert$ of states, the number $\lvert A^i\rvert$ of actions per agent, the number $\lvert\Omega^i\rvert$ of observations per agent and the discount factor γ. We report the value of the best-found FSC, its corresponding size as the number of memory nodes for each agent and the time it took to find this FSC. Note that InfJESP produces different type of FSCs (more akin to Moore machines), so the FSC sizes reported here are not easily comparable.

Benchmark	$\lvert S\rvert$	$\lvert Act\rvert$	$\lvert\Delta\rvert$
DPM	1,752	144,000	$1.6 \cdot 10^4$
Grid †	11,535	45,345	$6.5 \cdot 10^4$
Grid-10-4fsc	2,405	9,605	$6.5 \cdot 10^4$
Herman †	18,753	663,489	$3.1 \cdot 10^6$
Maze †	183	257	$9.4 \cdot 10^6$
Maze-5-3fsc	424	1,348	$7.9 \cdot 10^{25}$
Meet-grid-2fsc	9,216	577,000	10^8
Pole	9,241	13,000	$1.1 \cdot 10^{15}$
Refuel-06-2fsc	877	1,524	$1.1 \cdot 10^{41}$

Table 3: Statistics of the considered benchmarks for Q3: the model, the size of the underlying coloured MDP, and the size $\lvert\Delta\rvert$ of the design space. † denotes that the model is taken from (Andriushchenko et al., 2021).

driushchenko et al., 2021). Statistics of the benchmarks are listed in Table 3. This includes the size of the underlying coloured MDP and the size of the design space. Some of the models are parameterised; this includes the scalability of the model as well as the memory size of the FSC (if any).

The timeout for the evaluation of the engines was 30 minutes. On model *Pole*, there is a maximising expected reward specification which is not supported by the CEG engine (and therefore by the Hybrid engine), so their times are missing. The timeout for the previous implementation of PAYNT was set to 2 hours.

Comparison of the synthesis engines

Table 4 shows the performance of the synthesis approaches on selected benchmarks that demonstrate weak and strong features of the engines. It shows that CEG-based reasoning is very beneficial for the grid-world benchmarks (e.g. *Grid-10-4fsc*) primarily as this approach avoids computations on the (very large) MDP abstraction. Additionally, CEG is able to find small conflicts. On the remaining models, however, the CEG engine in isolation performs poorly and often times outs. A closer inspection shows that a large part of the design space is pruned using small conflicts, but in the remaining parts, the approach fails to find small conflicts, and thus, the later iterations are ineffective.

The AR engine with the new splitting strategy (its effect being evaluated later) performs well over the whole benchmark set. Its performance can be improved in some cases: the hybrid engine using the CE modulo bounds reduces the number of AR iterations significantly using only a small number of CEG iterations. For the *Refuel-06-2fsc* problem, the CEG integration is essential: 76k AR iterations are reduced to 61 AR and 2 CEG iterations, and thus, the runtime is reduced by about a factor of 500. We emphasise that although the hybrid engine is not always the fastest, it provides a robust synthesis method that is applicable without prior knowledge of the synthesis problem at hand.

Improvements to PAYNT

The new splitting strategy used in refinement leverages the scores estimating the impact of a given parameter assignment on the quality of the abstraction, see Section 3. Additionally, we implemented various optimisations to manipulate the design space and the CEs. Our experiments demonstrate that these enhancements significantly improve the performance of the previous version of PAYNT (Andriushchenko et al., 2021). The *Old Best* column in Table 4 lists the synthesis time for the best-performing synthesis engine in the previous version of PAYNT. The evaluation confirms that the improvements pay off in all considered benchmarks, and in some cases, they are essential for solving the synthesis problem. For benchmark *Maze*, e.g., it reduces the number of AR iterations by about three orders of magnitude, resulting in a speedup of over a factor of 3,000. The optimisations in CEG are beneficial for, e.g., the grid-based problems where the performance of AR is limited due to the large size of the MDP abstraction. These experiments clearly show that the presented improvements *significantly accelerate* the synthesis and enable to *solve more complicated synthesis problems*.

7. Related Work

7.1 Policy Synthesis for Models with Uncertainty

MDPs are important in both the model-checking community and the planning community (Klauck et al., 2020). To accommodate the different types of constraints on MDP policies, a plethora of extensions have been studied. Most prominently, POMDPs limit the information that a policy has access to, and decentralised (PO)MDPs address the aspect that a policy may be distributed over (independent) agents. Constrained MDPs (Altman, 1999) or multi-objective MDPs (Delgrange et al., 2020) allow expressing that policies must trade-off several specifications. All these models sacrifice the computational tractability of

	PAYNT						
	Hybrid		AR		CEG		
Benchmark	time	#iters	time	#iters	time	#iters	Old Best
DPM	138s	(3.5k, 781)	**110s**	3.7k	757s	15k	353s
Grid	3s	(35, 2.8k)	12s	7.5k	**1s**	1.2k	11s
Grid-10-4fsc	**22s**	(181, 7.6k)	64s	6.9k	**22s**	7.4k	3851s
Herman	**18s**	(11, 709)	61s	223	TO	-	48s
Maze	**<1s**	(139, 4)	**<1s**	143	TO	-	3028s
Maze-5-3fsc	38s	(12k, 25)	**37s**	12k	TO	-	TO
Meet-grid-2fsc	640s	(11k, 23k)	**532s**	15k	TO	-	TO
Pole	-	-	**3s**	146	-	-	5540s
Refuel-06-2fsc	**<1s**	(61, 2)	104s	76k	TO	-	TO

Table 4: Performance comparison of synthesis engines Hybrid, AR, and CEG. Hybrid is the symbiosis of abstraction refinement (AR) and counterexample generalisation (CEG). Hybrid iterations are tuples (AR iters, CEG iters). The maximising reward specification of benchmark *Pole* is not supported by CEG. Old Best refers to the time achieved by the best engine in the previous version of PAYNT (Andriushchenko et al., 2021).

the policy synthesis problem in MDPs in favour of expressivity. In our spirit, computing a policy from a large set of flexibly definable design spaces. Systems may be decentralised, policies may not be able to observe the complete system state, cost constraints may apply, and so forth. Adequate operational models for these systems exist in the form of *decentralised partially-observable MDPs* (DEC-POMDPs) (Oliehoek & Amato, 2016). The policy synthesis problem for these models is undecidable (Madani, Hanks, & Condon, 1999), and tool support to extract policies is scarce.

7.2 Parameter and Topology Synthesis

Parameter synthesis considers models with a fixed topology but with uncertain transition probabilities. The aim is to analyse how the behaviour of the given MC or MDP depends on the parameter values. Scalable approximate parameter synthesis treats identical parameters in different transitions independently (Češka et al., 2017; Quatmann et al., 2016) and has been implemented in the tools Storm (Dehnert et al., 2017) and PRISM (Kwiatkowska et al., 2011). Exact approaches construct rational functions for symbolic reachability probabilities (Daws, 2004; Hahn et al., 2011; Dehnert et al., 2015). *Topology synthesis* considers parameters that affect the topology of MCs akin to coloured MDPs. Finding a satisfying realisation w.r.t. some reachability property is NP-complete in the number of parameters (Chonev, 2019). The ProFeat (Chrszon et al., 2018) and QFLan (Vandin et al., 2018) tools model the MC family as an MDP and use off-the-shelf MDP model-checking algorithms for the analysis. The need for topology synthesis arises, e.g., while designing software product lines (Classen et al., 2012; Lanna et al., 2018). Search-based techniques that do not ensure complete exploration of the parameter space can be used to handle both these synthesis problems.

These include evolutionary techniques (Harman et al., 2012; Martens et al., 2010) and genetic algorithms (Gerasimou, Calinescu, & Tamburrelli, 2018).

7.3 Policy Synthesis for (Dec-)POMDPs

Widely used approaches for solving POMDPs build on the notion of *beliefs* (Smallwood & Sondik, 1973) that keep track of the probabilities of equally observable POMDP states. HSVI (Horak, Bosansky, & Chatterjee, 2018) and SARSOP (Kurniawati, Hsu, & Lee, 2008) are prominent belief-based policy synthesis approaches typically used for discounted properties. These approaches represent policies as a set of α-vectors. For policy search, alternatives are to search for randomised policies via gradient descent (Heck et al., 2022) or via convex optimisation (Junges et al., 2018; Cubuktepe et al., 2021). Recent approaches extract FSCs via deep reinforcement learning (Carr, Jansen, & Topcu, 2021). However, randomised policies limit predictability, which hampers testing and explainability. Deterministic FSCs are beneficial in this sense. Belief-based approaches for indefinite-horizon specification include (Norman, Parker, & Zou, 2017; Bork, Katoen, & Quatmann, 2022). Approaches based on mixed-integer linear programming (MILP) optimisation have also been proposed (Amato et al., 2010b; Kumar & Zilberstein, 2015). Finding policies for Dec-POMDPs is even more difficult problem due to scalability issues. Point-based algorithms such as FB-HSVI (Dibangoye et al., 2016) and PBVI-BB (MacDermed & Isbell, 2013) provide reasonable performance and scalability, but the deployment of the resulting policies is limited due to the lack of explainability compared to FSCs. Algorithms based on non-linear programming, such as MealyNLP (Amato et al., 2010b), struggle to scale to larger problems. Policy iteration producing randomised FSCs has also been proposed (Bernstein et al., 2009). Recent work on Dec-POMDPs includes: an extension of the multi-agent A* by introducing small steps in the search tree with recursive heuristics (Koops et al., 2023), Joint Equilibrium-based Search for Policies (JESP) with finding the best response policy for an agent when strategies of other agents are fixed called Inf-JESP (You et al., 2021) and Monte-Carlo search for equilibrium (You et al., 2023).

7.4 Constrained Optimisation for POMDPs

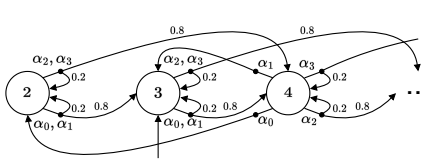
The need for optimisation in CPOMDPs arises naturally in many areas, and the task of finding near-optimal policies for these models is difficult (Wray & Czuprynski, 2022). The use of linear programs has been proposed as MILP formulations with dynamic programming updates (Isom et al., 2008), approximate linear programming (Poupart et al., 2015), iterative transition-based linear programming (Helmeczi et al., 2023) or ILP over constant-horizons (Khonji, Jasour, & Williams, 2019). Other proposed approaches include projected gradient ascent (Wray & Czuprynski, 2022) and point-based value iteration (Kim et al., 2011). Recent work in this area showed that solutions for CPOMDPs may have non-intuitive behaviour, which is not desirable for safety-critical applications. Recursively-constrained POMDPs, which impose history-based constraints, have been proposed as an alternative (Ho et al., 2023). Another alternative is considering multiple reward functions with preference ordering (Wray & Zilberstein, 2015). In this work, we address two problems related to constrained optimisation raised in (de Nijs et al., 2021): Firstly, our approach is able to provide formal guarantees on the given constraints even in the partially observable setting. Secondly, our approach can

```
const int Xmax = 6;
const int Xmin = 2;

hole int H1 in {3,4};
hole int H2 in {1,2};

module counter
    x : [2..6] init 3;

    x <= H1 -> 0.2: true + 0.8: (x'=min(x+H2, Xmax));
    x > H1  -> 1  : (x'=max(x-H2, Xmin));
endmodule
```



Colouring κ

$\kappa(4, \alpha_0) : H1 = 3, H2 = 2$
$\kappa(4, \alpha_1) : H1 = 3, H2 = 1$
$\kappa(4, \alpha_2) : H1 = 4, H2 = 1$
$\kappa(4, \alpha_3) : H1 = 4, H2 = 2$
$\kappa(2, \alpha_0) = \kappa(2, \alpha_1) : H2 = 1$
$\kappa(2, \alpha_2) = \kappa(2, \alpha_3) : H2 = 2$
$\kappa(3, \alpha_0) = \kappa(3, \alpha_1) : H2 = 1$
$\kappa(3, \alpha_2) = \kappa(3, \alpha_3) : H2 = 2$

Figure 5: Part of a coloured MDP encoding for the given program sketch. The states describe the value of variable x.

handle even undiscounted specifications while keeping good performance, dropping the usage of often impractical discounted constraints.

8. Conclusions

We presented a novel synthesis framework for systems operating in uncertain environments and limited observability. In contrast to dedicated synthesis methods, it supports a wide range of synthesis domains, including sketching of probabilistic programs and solving (decentralised) POMDPs. It also supports complex (undiscounted) indefinite-horizon specifications as well as constrained optimisation objectives. The framework is implemented in the tool PAYNT. Experiments provide promising results; for some synthesis problems, it outperforms existing, dedicated synthesis techniques. The modular framework can further benefit from other synthesis methods such as belief-based exploration (Andriushchenko et al., 2023).

Acknowledgments

This work has been supported by the Czech Science Foundation grant GA23-06963S (VESCAA), the IGA VUT project FIT-S-23-8151, the ERC Advanced Grant FRAPPANT (787914), and the NWO VENI Grant ProMiSe (222.147).

Appendix A. Translating Program Sketches into Coloured MDP

A PRISM program sketch contains a finite set of holes, each hole with a finite domain. The holes can be either in guards or updates of commands. A single command can include several holes, and a single hole can be used in several commands. The left part of Fig. 5 illustrates a simple PRISM sketch.

A sketch realisation r is a complete mapping from holes onto their domains. It defines a complete program with the semantics given by a finite-state MC M^r. The goal is to find a realisation r such that M^r satisfies the specification. In (Češka et al., 2021), we formalised the sketches of probabilistic programs using families of MCs over a finite state of parameters H where each parameter has a finite domain. Intuitively, states are tuples that encode program-variable valuations and transitions are given by a parameterised transition probability function $\mathcal{B} : S_1 \times S_2 \times \ldots S_n \mapsto \mathit{Distr}((S_1 \cup H) \times (S_2 \cup H) \times \ldots (S_n \cup H))$ together with domain constraints ensuring that each family member is well-formed.

Such families can be compactly represented by a coloured $\mathcal{M}$. Intuitively, the underlying MDP encodes the state space and each action available in the state $(s_1, s_2, \ldots, s_n)$ encodes

one parametrisation of the transition function $\mathcal{B}$. The colouring $\kappa(s, \alpha)$ coincides with the valuation of the parameters given by the action α. The right part of Fig. 5 illustrates a part of the coloured MDP for the sketch on the left.

Appendix B. Translating POMDPs into Coloured MDP

POMDPs extend MDP with a finite set Z of *observations* and an *observation function* O. For simplicity, we assume POMDPs with a deterministic observation function $O\colon S \to Z$ that maps each state to an observation $z \in Z$[10] A memoryless policy σ is observation based if $O(s) = O(s') \implies \sigma(s) = \sigma(s')$ for all $s, s' \in S$. For a given specification, the synthesis of memoryless observation-based policies can be encoded using $\mathcal{M}$ where $H = \{h_z \mid z \in Z\}$ and $\kappa(s, \alpha)(h_z) = \alpha$ if $z = O(s)$, $\bot$ otherwise.

Memoryless policies are typically insufficient for POMDPs; thus, a more general class of observation-based policies mapping a sequence of observations to an action is considered (Kaelbling, Littman, & Cassandra, 1998). Using *finite-state controllers* (FSCs) (Hansen, 1998) is one of the prominent approaches to represent such policies. There exist various variants of the FSCs. We focus on deterministic Mealy machines given by 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. Imposing k-FSC F onto POMDP $M = (S, s_0, Act, \mathcal{P})$ with observation function O yields the *induced* Markov chain $\mathcal{M}^F = (S \times N, (s_0, n_0), \mathcal{P}^F)$ where $\mathcal{P}^F((s, n), (s', n')) = \mathcal{P}(s, \gamma(n, z), s') \cdot [\delta(n, z) = n']$[11] for $z = O(s)$.

For a fixed k, the goal is to find a FSC F such that $\mathcal{M}^F$ satisfies the given specification. Such synthesis problem can be encoded using $\mathcal{M} = (M, \Delta, \kappa)$ where $M = (S \times N, (s_0, n_0), Act \times N, \mathcal{P})$ with $\mathcal{P}((s, n), (\alpha, n'), (s', n')) = \mathcal{P}(s, \alpha, s')$ for all $n, n' \in N$. The design space Δ is given by the parameters $H = \{(h_z, n) \mid z \in Z, n \in N\}$ and domains $D((h_z, n)) = Act(s) \times N$ for $z = O(s)$. To encode the dependencies in H, we define $\kappa((s, n), (\alpha, n'))(h_z, n) = (\alpha, n')$ if $z = O(s)$, $\bot$ otherwise.

This construction can be straightforwardly extended to capture the synthesis of FSCs in decentralised POMDPs where the goal is to synthesise a set of independent local FSCs that together supply the joint controller. Note that each local FSC chooses an action based purely on its local observation and memory. For two agents, the synthesis of two k-FSCs F_1 and F_2 can be encoded using a coloured $\mathcal{M}$ defined over the state space $S \times N \times N$, the action space $Act \times Act \times N \times N$, where parameters $H = H_1 \cup H_2$ corresponding to the parameters in F_1 and F_2 respectively, and for $i \in \{1, 2\}$, $\kappa((s, n_1, n_2), (\alpha_1, \alpha_2, n_1', n_2'))(h_z, n_i) = (\alpha_i, n_i')$ if $(h_z, n_i) \in H_i$ and $z = O_i(s)$ (agent i observes z in s), $\bot$ otherwise.

References

Ábrahám, E., Becker, B., Dehnert, C., Jansen, N., Katoen, J.-P., & Wimmer, R. (2014).

10. Observation functions resulting in a distribution over observations can be encoded by deterministic observation functions at the expense of a polynomial blow-up (Chatterjee, Chmelík, Gupta, & Kanodia, 2016).
11. Iverson-brackets: $[x] = 1$ if predicate x is true, 0 otherwise.

Counterexample generation for discrete-time Markov models: An introductory survey. In *SFM*, pp. 65–121. Springer.

Altman, E. (1999). *Constrained Markov Decision Processes*. Stochastic Modeling Series. Taylor & Francis.

Alur, R., et al. (2015). Syntax-guided synthesis. In *Dependable Software Systems Engineering*, Vol. 40 of *NATO Science for Peace and Security Series*, pp. 1–25. IOS Press.

Amato, C., Bernstein, D. S., & Zilberstein, S. (2010a). Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *Auton. Agents Multi-Agent Syst.*, *21*(3), 293–320.

Amato, C., Bonet, B., & Zilberstein, S. (2010b). Finite-state controllers based on Mealy machines for centralized and decentralized POMDPs. In *AAAI*, pp. 1052–1058. AAAI Press.

Andriushchenko, R., Bork, A., Češka, M., Junges, S., Katoen, J.-P., & Macák, F. (2023). Search and explore: Symbiotic policy synthesis in POMDPs. In *CAV*, pp. 113–135. Springer.

Andriushchenko, R., Češka, M., Junges, S., & Katoen, J.-P. (2021). Inductive synthesis for probabilistic programs reaches new horizons. In *TACAS*, pp. 191–209. Springer.

Andriushchenko, R., Češka, M., Junges, S., & Katoen, J.-P. (2022). Inductive synthesis of finite-state controllers for POMDPs. In *UAI*, pp. 85–95. PMRL.

Andriushchenko, R., Češka, M., Junges, S., Katoen, J.-P., & Macák, F. (2024). Artifact supplement for 'an oracle-guided approach to constrained controller synthesis under uncertainty'. https://doi.org/10.5281/zenodo.11401350.

Andriushchenko, R., Češka, M., Junges, S., Katoen, J.-P., & Stupinský, Š. (2021). PAYNT: a tool for inductive synthesis of probabilistic programs. In *CAV*, pp. 856–869. Springer.

Baier, C., de Alfaro, L., Forejt, V., & Kwiatkowska, M. (2018). Model checking probabilistic systems. In *Handbook of Model Checking*, pp. 963–999. Springer.

Barbosa, H., Barrett, C., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., & Zohar, Y. (2022). cvc5: A versatile and industrial-strength SMT solver. In *TACAS*, pp. 415–442. Springer.

Benini, L., Bogliolo, A., Paleologo, G. A., & De Micheli, G. (1999). Policy optimization for dynamic power management. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, *18*(6), 813–833.

Bernstein, D. S., Amato, C., Hansen, E. A., & Zilberstein, S. (2009). Policy iteration for decentralized control of Markov decision processes. *J. Artif. Int. Res.*, *34*(1).

Bertsekas, D. P. (2005). *Dynamic Programming and Optimal Control* (3rd edition)., Vol. I. Athena Scientific.

Bork, A., Katoen, J.-P., & Quatmann, T. (2022). Under-approximating expected total rewards in POMDPs. In *TACAS (2)*, pp. 22–40. Springer.

Carr, S., Jansen, N., & Topcu, U. (2021). Task-aware verifiable RNN-based policies for partially observable Markov decision processes. *J. Artif. Intell. Res.*, *72*, 819–847.

Češka, M., Dannenberg, F., Paoletti, N., Kwiatkowska, M., & Brim, L. (2017). Precise parameter synthesis for stochastic biochemical systems. *Acta Inf.*, *54*(6), 589–623.

Češka, M., Hensel, C., Junges, S., & Katoen, J.-P. (2019a). Counterexample-driven synthesis for probabilistic program sketches. In *FM*, LNCS, pp. 101–120. Springer.

Češka, M., Jansen, N., Junges, S., & Katoen, J.-P. (2019b). Shepherding hordes of Markov chains. In *TACAS (2)*, pp. 172–190. Springer.

Chatterjee, K., Chmelík, M., Gupta, R., & Kanodia, A. (2016). Optimal cost almost-sure reachability in POMDPs. *Artif. Intell.*, *234*, 26–48.

Chonev, V. (2019). Reachability in augmented interval Markov chains. In *RP*, pp. 79–92. Springer.

Chrszon, P., Dubslaff, C., Klüppelholz, S., & Baier, C. (2018). ProFeat: feature-oriented engineering for family-based probabilistic model checking. *Formal Asp. Comput.*, *30*(1), 45–75.

Classen, A., Cordy, M., Heymans, P., Legay, A., & Schobbens, P.-Y. (2012). Model checking software product lines with SNIP. *Int. J. on Softw. Tools for Technol. Transf.*, *14*, 589–612.

Cropper, A., & Dumančić, S. (2022). Inductive logic programming at 30: a new introduction. *J. Artif. Int. Res.*, *74*, 765–850.

Cubuktepe, M., Jansen, N., Junges, S., Marandi, A., Suilen, M., & Topcu, U. (2021). Robust finite-state controllers for uncertain POMDPs. In *AAAI*, pp. 11792–11800. AAAI Press.

Daws, C. (2004). Symbolic and parametric model checking of discrete-time Markov chains. In *ICTAC*, pp. 280–294. Springer.

de Moura, L., & Bjørner, N. (2008). Z3: An efficient SMT solver. In *TACAS*, pp. 337–340. Springer.

de Nijs, F., Walraven, E., De Weerdt, M., & Spaan, M. (2021). Constrained multiagent Markov decision processes: a taxonomy of problems and algorithms. *J. Artif. Int. Res.*, *70*.

Dehnert, C., Jansen, N., Wimmer, R., Ábrahám, E., & Katoen, J.-P. (2014). Fast debugging of PRISM models. In *ATVA*, pp. 146–162. Springer.

Dehnert, C., Junges, S., Jansen, N., Corzilius, F., Volk, M., Bruintjes, H., Katoen, J.-P., & Ábrahám, E. (2015). PROPhESY: A PRObabilistic ParamEter SYNnthesis Tool. In *CAV*, pp. 214–231. Springer.

Dehnert, C., Junges, S., Katoen, J.-P., & Volk, M. (2017). A Storm is coming: A modern probabilistic model checker. In *CAV*, pp. 592–600. Springer.

Delgrange, F., Katoen, J.-P., Quatmann, T., & Randour, M. (2020). Simple strategies in multi-objective MDPs. In *TACAS*, pp. 346–364. Springer.

Dibangoye, J. S., Amato, C., & Doniec, A. (2012). Scaling up decentralized MDPs through heuristic search. In *UAI*, pp. 217–226. AUAI Press.

Dibangoye, J. S., Amato, C., Buffet, O., & Charpillet, F. (2016). Optimally solving Dec-POMDPs as continuous-state MDPs. *J. Artif. Int. Res.*, *55*(1).

Ferrer-Mestres, J., Dietterich, T. G., Buffet, O., & Chades, I. (2021). K-N-MOMDPs: towards interpretable solutions for adaptive management. In *AAAI*, pp. 14775–14784. AAAI Press.

Forejt, V., Kwiatkowska, M. Z., & Parker, D. (2012). Pareto curves for probabilistic model checking. In *ATVA*, pp. 317–332. Springer.

Funke, F., Jantsch, S., & Baier, C. (2020). Farkas certificates and minimal witnesses for probabilistic reachability constraints. In *TACAS (1)*, pp. 324–345. Springer.

Gerasimou, S., Tamburrelli, G., & Calinescu, R. (2015). Search-based synthesis of probabilistic models for quality-of-service software engineering (T). In *ASE*, pp. 319–330. IEEE Computer Society.

Gerasimou, S., Calinescu, R., & Tamburrelli, G. (2018). Synthesis of probabilistic models for quality-of-service software engineering. *Autom. Softw. Eng.*, *25*(4), 785–831.

Hahn, E. M., Hermanns, H., & Zhang, L. (2011). Probabilistic reachability for parametric Markov models. *Int. J. on Softw. Tools for Technol. Transf.*, *13*(1), 3–19.

Hansen, E. A. (1998). Solving POMDPs by searching in policy space. In *UAI*, pp. 211–219. Morgan Kaufmann.

Harman, M., Mansouri, S. A., & Zhang, Y. (2012). Search-based software engineering: Trends, techniques and applications. *ACM Comp. Surveys*, *45*(1), 1–61.

Hauskrecht, M. (1997). Incremental methods for computing bounds in partially observable Markov decision processes. In *AAAI/IAAI*, pp. 734–739. AAAI Press.

Hayes, C. F., Radulescu, R., Bargiacchi, E., et al. (2023). A brief guide to multi-objective reinforcement learning and planning. In *AAMAS*, pp. 1988–1990. ACM.

Heck, L., Spel, J., Junges, S., Moerman, J., & Katoen, J. (2022). Gradient-descent for randomized controllers under partial observability. In *VMCAI*, pp. 127–150. Springer.

Helmeczi, R. K., Kavaklioglu, C., & Cevik, M. (2023). Linear programming-based solution methods for constrained partially observable Markov decision processes. *Applied Intelligence*, *53*(19).

Herman, T. (1990). Probabilistic self-stabilization. *Inf. Process. Lett.*, *35*(2), 63–67.

Ho, Q. H., Becker, T., Kraske, B., Laouar, Z., Feather, M. S., Rossi, F., Lahijanian, M., & Sunberg, Z. N. (2023). Recursively-Constrained partially observable Markov decision processes..

Horak, K., Bosansky, B., & Chatterjee, K. (2018). Goal-HSVI: Heuristic search value iteration for Goal POMDPs. In *IJCAI*, pp. 4764–4770. AAAI Press.

Isom, J. D., Meyn, S. P., & Braatz, R. D. (2008). Piecewise linear dynamic programming for Constrained POMDPs. In *AAAI*, pp. 291–296.

Jha, S., Gulwani, S., Seshia, S. A., & Tiwari, A. (2010). Oracle-guided component-based program synthesis. In *ICSE*, pp. 215–224. ACM.

Junges, S. (2020). *Parameter Synthesis in Markov Models*. Ph.D. thesis, RWTH Aachen University, Germany.

Junges, S., Jansen, N., Wimmer, R., Quatmann, T., Winterer, L., Katoen, J.-P., & Becker, B. (2018). Finite-state controllers of POMDPs via parameter synthesis. In *UAI*, pp. 519–529. AUAI Press.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artif. Intell.*, *101*(1-2), 99–134.

Khonji, M., Jasour, A., & Williams, B. C. (2019). Approximability of constant-horizon Constrained POMDP. In *IJCAI*, pp. 5583–5590. AAAI Press.

Kim, D., Lee, J., Kim, K.-E., & Poupart, P. (2011). Point-based value iteration for Constrained POMDPs. In *IJCAI*, pp. 1968–1974. AAAI Press.

Klauck, M., Steinmetz, M., Hoffmann, J., & Hermanns, H. (2020). Bridging the gap between probabilistic model checking and probabilistic planning: Survey, compilations, and empirical comparison. *J. Artif. Int. Res.*, *68*.

Kolobov, A., Mausam, Weld, D. S., & Geffner, H. (2011). Heuristic search for generalized stochastic shortest path MDPs. In *ICAPS*, p. 130–137. AAAI Press.

Koops, W., Jansen, N., Junges, S., & Simão, T. D. (2023). Recursive small-step multi-agent A* for Dec-POMDPs. In *IJCAI*, pp. 5402–5410.

Kumar, A., & Zilberstein, S. (2015). History-based controller design and optimization for partially observable MDPs. In *ICAPS*, pp. 156–164. AAAI Press.

Kurniawati, H., Hsu, D., & Lee, W. S. (2008). SARSOP: Efficient point-based POMDP planning by approximating optimally reachable belief spaces. In *Robotics: Science and Systems*. MIT Press.

Kwiatkowska, M., Norman, G., & Parker, D. (2012). Probabilistic verification of Herman's self-stabilisation algorithm. *Form. Asp. Comput.*, *24*(4), 661–670.

Kwiatkowska, M., Norman, G., & Parker, D. (2011). PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, pp. 585–591. Springer.

Lanna, A., Castro, T., Alves, V., Rodrigues, G., Schobbens, P.-Y., & Apel, S. (2018). Feature-family-based reliability analysis of software product lines. *Inf. and Softw. Technol.*, *94*, 59–81.

Littman, M. L., Cassandra, A. R., & Kaelbling, L. P. (1995). Learning policies for partially observable environments: Scaling up. In *Machine Learning Proceedings*.

MacDermed, L. C., & Isbell, C. L. (2013). Point based value iteration with optimal belief compression for Dec-POMDPs. In *NIPS*, Vol. 26.

Madani, O., Hanks, S., & Condon, A. (1999). On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *AAAI/IAAI*, pp. 541–548. AAAI.

Martens, A., Koziolek, H., Becker, S., & Reussner, R. (2010). Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In *WOSP/SIPEW*, pp. 105–116. ACM.

Mertens, H., Katoen, J.-P., Quatmann, T., & Winkler, T. (2024). Accurately computing expected visiting times and stationary distributions in Markov chains. In *TACAS*, pp. 237–257.

Norman, G., Parker, D., & Zou, X. (2017). Verification and control of partially observable probabilistic systems. *Real-Time Syst.*, *53*(3), 354–402.

Oliehoek, F. A., & Amato, C. (2016). *A Concise Introduction to Decentralized POMDPs*. Springer.

Osera, P.-M., & Zdancewic, S. (2015). Type-and-example-directed program synthesis. In *PLDI '15*. ACM.

Poupart, P., Malhotra, A., Pei, P., Kim, K.-E., Goh, B., & Bowling, M. (2015). Approximate linear programming for constrained partially observable Markov decision processes. In *AAAI*, pp. 3342–3348. AAAI Press.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley.

Quatmann, T., Dehnert, C., Jansen, N., Junges, S., & Katoen, J.-P. (2016). Parameter synthesis for Markov models: Faster than ever. In *ATVA*, pp. 50–67. Springer.

Quatmann, T., Jansen, N., Dehnert, C., Wimmer, R., Ábrahám, E., Katoen, J.-P., & Becker, B. (2015). Counterexamples for expected rewards. In *FM*, pp. 435–452. Springer.

Smallwood, R. D., & Sondik, E. J. (1973). The optimal control of partially observable Markov processes over a finite horizon. *Oper. Res.*, *21*(5), 1071–1088.

Solar-Lezama, A. (2013). Program sketching. *Int. J. Softw. Tools Technol. Transf.*, *15*(5-6), 475–495.

Solar-Lezama, A., Rabbah, R., Bodík, R., & Ebcioğlu, K. (2005). Programming by sketching for bit-streaming programs. In *PLDI*, pp. 281–294. ACM.

Solar-Lezama, A., Tancau, L., Bodik, R., Seshia, S., & Saraswat, V. (2006). Combinatorial sketching for finite programs. In *ASPLOS'06*. ACM.

Vandin, A., ter Beek, M. H., Legay, A., & Lluch-Lafuente, A. (2018). QFLan: A tool for the quantitative analysis of highly reconfigurable systems. In *FM*, pp. 329–337. Springer.

Češka, M., Hensel, C., Junges, S., & Katoen, J.-P. (2021). Counterexample-guided inductive synthesis for probabilistic systems. *Form. Asp. Comput.*, *33*(4–5), 637–667.

Vos, D., & Verwer, S. (2023). Optimal decision tree policies for Markov decision processes. In *IJCAI*, pp. 5457–5465. ijcai.org.

Wimmer, R., Jansen, N., Ábrahám, E., Katoen, J., & Becker, B. (2014). Minimal counterexamples for linear-time probabilistic verification. *Theor. Comput. Sci.*, *549*, 61–100.

Wimmer, R., Jansen, N., Vorpahl, A., Ábrahám, E., Katoen, J.-P., & Becker, B. (2015). High-level counterexamples for probabilistic automata. *Log. Methods Comput. Sci.*, *11*(1).

Wray, K. H., & Czuprynski, K. (2022). Scalable gradient ascent for controllers in Constrained POMDPs. In *ICRA*, pp. 9085–9091. IEEE Press.

Wray, K. H., & Zilberstein, S. (2015). Multi-objective POMDPs with lexicographic reward preferences. In *IJCAI*, pp. 3418–3424. AAAI Press.

You, Y., Thomas, V., Colas, F., & Buffet, O. (2021). Solving infinite-horizon Dec-POMDPs using finite state controllers within JESP. In *ICTAI*, pp. 427–434. IEEE.

You, Y., Thomas, V., Colas, F., & Buffet, O. (2023). Monte-Carlo search for an equilibrium in Dec-POMDPs. In *UAI*, pp. 2444–2453. PMLR.