

# Synthesizing Safe Policies under Probabilistic Constraints with Reinforcement Learning and Bayesian Model Checking

Lenz Belzner and Martin Wirsing

**Draft.** Final Version: Lenz Belzner and Martin Wirsing. Synthesizing Safe Policies under Probabilistic Constraints with Reinforcement Learning and Bayesian Model Checking. Science of Computer Programming, 2021, 102620, ISSN 0167-6423.

## Abstract

We propose to leverage epistemic uncertainty about constraint satisfaction of a reinforcement learner in safety critical domains. We introduce a framework for specification of requirements for reinforcement learners in constrained settings, including confidence about results. We show that an agent’s confidence in constraint satisfaction provides a useful signal for balancing optimization and safety in the learning process.

## 1 Introduction

Reinforcement learning enables agents to automatically learn policies maximizing a given reward signal. Recent developments combining reinforcement learning with deep learning have had great success in tackling more and more complex domains, such as learning to play video games based on visual input or enabling automated real-time scheduling in production systems [6, 62].

Reinforcement learning also provides valuable solutions for systems operating in non-deterministic and partially known environments, such as autonomous systems, socio-technical systems and collective adaptive systems (see e.g. [16, 37, 19, 51]). However, it is often difficult to ensure the quality and the correctness of reinforcement learning solutions [58, 5]. In many applications, learning is focusing on achieving and optimizing system behavior but not on guaranteeing the safety of the system (see e.g. also [16, 37, 19]).

Optimizing for both functional effectiveness and system safety at the same time poses a fundamental challenge: If maximizing return and constraint satisfaction are interfering, what should the learner optimize in a given situation? That is, besides the fundamental dilemma of exploration and exploitation, the learner now faces an additional choice to be made: When to optimize return,

and when to optimize feasibility in case it is not possible to optimize both at the same time?

This fundamental question has been addressed in many related works, which can be categorized in three broad classes: Learning safe behavior using a given or learned model of the environment [42, 34, 35], learning shields in addition to a policy assuring that only safe actions are executed [3, 17, 7, 40] or reward-shaping methods that try to balance optimization of return and costs incurred by constraint violation [56, 20, 21, 28].

Typically satisfaction probability is estimated via maximum likelihood, ignoring the learner’s uncertainty about intermediate estimates in the empirical learning process. We argue that when using reinforcement learning for policy synthesis, additionally requiring a specification of *confidence* in the result is of high importance as reinforcement learning is an empirical technique relying on finite data points. We also argue that using a learner’s confidence in requirement satisfaction provides a highly informative signal for effectively exploring the Pareto front of return optimization and safety.

To this end, we propose *Policy Synthesis under probabilistic Constraints* (PSyCo), a systematic method for specifying and implementing agents that shape rewards dynamically over the learning process based on their confidence in requirement satisfaction. The basic idea is to emphasize return optimization when the learner is confident, and to focus on satisfying given constraints otherwise. This enables to explicitly distinguish requirements wrt. aleatoric uncertainty that is inherent to the domain, and epistemic uncertainty arising from an agent’s learning process based on limited observations.

For implementing PSyCo’s abstract design we propose *Safe Neural Evolutionary Strategies* (SNES) for model-free learning of safe policies wrt. given finite-horizon specifications. SNES leverages Bayesian model checking while learning to adjust the Lagrangian of a constrained optimization problem according to the learner’s current confidence in specification satisfaction. The model-free formulation allows to synthesize safe policies in domains where only a generative model of the environment is available, enabling learning when the model is unknown to the agent or where analytical computation of solutions wrt. transition dynamics is intractable for closed form representations. Also, the model-free formulation enables fast inference (i.e. action selection) at runtime, rendering our approach feasible for real-time application domains.

The paper makes the following contributions.

- *Policy Synthesis under probabilistic Constraints* (PSyCo), a systematic method for shaping rewards dynamically over the learning process based on the learner’s confidence in requirement satisfaction. PSyCo accounts for empirical policy synthesis and verification based on finite observations by including requirements on confidence in synthesized results.
- *Safe Neural Evolutionary Strategies* (SNES) for learning safe policies under probabilistic constraints. SNES leverages online Bayesian model checking to obtain estimates of constraint satisfaction probability and a confidence in this estimate.

- We empirically evaluate SNES showing it is able to synthesize policies that satisfy probabilistic constraints with a required confidence. Code for our experiments is available at <https://github.com/lenzbelzner/psyco>.

The paper is structured as follows: In Section 2 we introduce the PSyCo method. Section 3 describes safe policy synthesis with the Safe Neural Evolutionary Strategy SNES. In Section 4 we present the results of two experiments, the so-called Particle Dance and Obstacle Run case studies. Sections 5 and 6 discuss related work and the limitations of our approach. Finally, Section 7 gives a short summary of PSyCo and addresses further work.

## 2 The PSyCo Method for Safe Policy Synthesis under Probabilistic Constraints

Our approach to safe policy synthesis comprises three phases: System specification as constrained Markov decision process with goal-oriented requirements, system design and implementation via safe policy synthesis, and verification by Bayesian model checking. As we will see, we use Bayesian model checking in two ways: To guide the learning process towards feasible solutions, and to verify synthesized policies.

### 2.1 PSyCo Overview

PSyCo comprises the four components.

- A domain specification given by a Markov decision process. We emphasize that our approach is model-free, therefore only requiring a generative model of the transition dynamics. The model-free formulation allows to synthesize safe policies when the dynamics model is unknown or where analytical computation of solutions wrt. transition dynamics is intractable for closed form representations.
- A set of goal-oriented requirements including optimization goals and probabilistic safety constraints. We restrict our further discussion to a single optimization goal and a single safety constraint for the sake of simplicity. We think that extending our results to sets of constraints is straightforward. We derive a cost function from requirements, effectively turning the domain MDP into a constrained Markov decision process (CMDP).
- A safe reinforcement learning algorithm  $L$  yielding the parameters of a policy.
- A verification algorithm  $V$  to check constraint satisfaction of the learned policy in the given CMDP.

PSyCo leverages a learning algorithm  $L$  for synthesizing safe policies wrt. rewards, costs and constraints (i.e. optimizing goals and probabilistic constraints), and a verification algorithm  $V$  for statistically verifying synthesized policies.

We optimize the parameters  $\theta \in \Theta$  of the policy wrt. rewards and costs of the derived constrained CMDP  $m \in M$  with the safe reinforcement learning algorithm  $L$ , taking the given probabilistic requirement  $\varphi \in \Phi$  into account.

$$L : M \times \Phi \rightarrow \Theta \tag{1}$$

We verify the optimized parameters of the synthesized policy wrt. the given CMDP and the constraint specification.

$$V : M \times \Phi \times \Theta \rightarrow \mathbb{B} \tag{2}$$

Given a CMDP  $m$  and a constraint  $\varphi$ , PSyCo works by learning and verifying a policy as follows (where  $\theta = L(m, \varphi)$ ).

$$\text{PSyCo} : M \times \Phi \rightarrow \mathbb{B} \tag{3}$$

$$\text{PSyCo}(m, \varphi) = V(m, \varphi, L(m, \varphi)) = V(m, \varphi, \theta) \tag{4}$$

## 2.2 System Specification: Constrained Markov Decision Processes and Goal-Oriented Requirements

**Domain Specification as Constrained MDP** The specification for a particular domain is given by a set  $S$  of states, a distribution over initial states  $\rho : p(S)$ , a set  $A$  of actions, and a reward function  $R : S \times A \times S \rightarrow \mathbb{R}$  encoding optimization goals for the agent.

We assume the domain has probabilistic transition dynamics  $T : p(S|S, A)$ . Note that  $T$  may be *unknown* to the agent.

A specification comprises a set of requirements or constraints given in a bounded variant of PCTL that we define below. We also define a way to transform these constraints into a cost function  $C : S \times A \times S \rightarrow \mathbb{R}$ .  $(S, A, T, R, C, \rho)$  constitutes a constrained Markov decision process (CMDP) [4].

An episode  $\vec{e} \in E$  is a finite sequence of transitions  $(s_i, a_i, s_{i+1}, r_i, c_i)$ ,  $s_i, s_{i+1} \in S, a_i \in A, r_i = R(s_i, a_i, s_{i+1}), c_i = C(s_i, a_i, s_{i+1})$  in the CMDP. The sequence  $\vec{s} = s_0, \dots, s_{n-1}$  denotes the path  $s(\vec{e})$  of that episode and  $Path_n$  denotes the set of all paths of length  $n$ .

In a CMDP, the task is to synthesize a deterministic, memoryless policy  $\pi : S \rightarrow A$  that maximizes reward while minimizing costs. We will formally specify our particular task after introducing our safety specifications and their transformation into cost functions below.

We emphasize that our approach is model-free, therefore only requiring a generative model of the transition dynamics  $T$ . The model-free formulation allows us to synthesize safe policies when the dynamics model is unknown or where analytical computation of solutions wrt. transition dynamics is intractable for closed form representations. Note that as we only rely on sampling the domain for policy synthesis in the following, even partial observations and a generative model of the reward function are sufficient for our approach to work.

Note that CMDPs are not restricted to a single cost function in general, however in this paper we restrict ourselves to a single cost function for sake of simplicity. We think that our results could be extended to sets of cost functions straightforwardly.

**Requirements** We consider two kinds of goals: *optimization goals* and *safety constraints*. Optimization goals are soft constraints and maximize an objective function, constraints are behavioral goals which impact the possible behaviors of the system, similar to e.g. maintain goals (which restrict the behavior of the system) and achieve goals (which generate behavior), see KAOS [24]. In our setting we relate the optimization goal with the rewards of the MDP and require it to maximize the return:

$$\text{Goal Optimize Return : } \max \mathbb{E}(\mathcal{R}) \quad (5)$$

where the return  $\mathcal{R}$  is the cumulative sum of rewards  $\mathcal{R} = \sum_{i=0}^{|\vec{e}|-1} r_i$  in an episode  $\vec{e}$  and  $\mathbb{E}(\mathcal{R})$  denotes the expectation of the return.

A suitable logic to express safety constraints in our setup is probabilistic computation tree logic (PCTL), allowing to specify constraints on satisfaction probabilities as well as bounding costs that may arise in system execution [55, 11]. We interpret PCTL formulas as bounded by the length of an episode  $n \in \mathbb{N}$  and consider typically formulas of the form

$$\mathbb{P}_{\geq p_{\text{req}}} (\Box \phi) \quad (6)$$

where  $\Box \phi$  is a so-called path formula and  $\phi$  is a propositional state formula built according to the syntax for CTL state formulas ( $a$  is an atomic state proposition,  $\phi_1, \phi_2$  are state formulas).

$$\phi = \text{true} \mid a \mid \phi_1 \wedge \phi_2 \mid \neg \phi \quad (7)$$

Formula 6 states that with at least probability  $p_{\text{req}}$ , the constraint  $\phi$  holds in every state of the path of the episode (i.e.  $\forall k, 0 \leq k \leq n : s_k \models \phi$ ). For the formal semantics and a more general treatment of other formulas in PCTL, we refer the reader to the appendix A.

Given that reinforcement learning is an empirical approach for synthesizing policies, it is reasonable to require a confidence in any synthesized result. Therefore we extend our constraints by an explicit operator  $\mathbb{C}_{c_{\text{req}}}$ , requiring the rate of false positive and false negative verification results to be bounded by  $1 - c_{\text{req}}$ . This allows to distinguish between aleatoric (i.e. domain-inherent, irreducible) and epistemic (i.e. agent-inherent, reducible) uncertainty in the specification of requirements. PSyCo enables to specify requirements for aleatoric uncertainty by bounding the probability of a constraint violation and epistemic uncertainty by specifying a bound on confidence. We denote our requirements wrt. constraint violation probability and agent confidence by

$$\mathbb{P}_{\geq p_{\text{req}}} (\Box \phi) \text{ and } \mathbb{C}_{\geq c_{\text{req}}} \quad (8)$$

requiring that  $\Box\phi$  holds with probability of at least  $p_{req}$  and confidence of at least  $c_{req}$  ( $p_{req}, c_{req} \in [0, 1]$ ).

Let us consider how to derive a cost function  $C$  from a given constraint. For any propositional state formula  $\phi$  we define the cost function  $C : S \times A \times S \rightarrow \mathbb{R}$  such that

$$C_\phi(s, a, s') \begin{cases} = 0 & \text{if } s' \models \phi \\ > 0 & \text{otherwise} \end{cases} \quad (9)$$

Note that another value of  $C$  for a violation of  $\phi$  could be given by a more general function of the post-state  $s'$ , giving the possibility to quantify the severeness of a present violation.

The cumulative cost  $\mathcal{C}$  of an episode  $\vec{e}$  of length  $n$  with path  $\vec{s}$  for a safety constraint  $\Box\phi$  is defined by the sum of violations of  $\phi$  over the course of the episode.

$$\mathcal{C}_{\Box\phi, \vec{s}} = c_\phi(s_0) + \sum_{i=0}^{n-1} C_\phi(s_i, a, s_{i+1}) \quad (10)$$

where the cost  $c_\phi(s_0)$  of the initial state of an episode is computed with the function  $c_\phi : S \rightarrow \mathbb{R}$  defined by

$$c_\phi(s) = \begin{cases} 0 & \text{if } s \models \phi \\ 1 & \text{otherwise} \end{cases} \quad (11)$$

In this way, we get a measure of how often an violation actually happened. This provides a more fine-grained learning signal to the learning agent than a strictly binary reward for satisfaction or violation.

Also note that the cumulative cost of an episode is zero exactly if no state in the episode violates  $\phi$ . Then the relation of a given safety constraint and the cumulative cost of an episode  $\vec{e}$  with path  $\vec{s}$  is given by

$$\vec{e} \models \Box\phi \text{ iff } \mathcal{C}_{\Box\phi, \vec{s}} = 0 \quad (12)$$

For more general path formulas and details see appendix A and B.

### 2.3 Abstract Design: Safe Reinforcement Learning Algorithm L

The task defined by a PSyCo specification is to synthesize a deterministic, memoryless policy  $\pi : S \rightarrow A$  that optimizes the following constrained optimization problem for bounded episodes.

$$\max \mathbb{E}(\mathcal{R}) \text{ s.t. } \mathbb{P}_{\geq p_{req}}(\mathcal{C}_\phi = 0) \text{ and } \mathbb{C}_{\geq c_{req}} \quad (13)$$

We now discuss the learning algorithm L for synthesizing a policy optimizing its parameters wrt. this constrained optimization problem.

We denote executing a policy parameterized by  $\theta \in \Theta$  in a CMDP  $m \in M$  as follows, yielding a distribution over episodes.

$$m : p(E|\Theta) \tag{14}$$

We sample episodes from the distribution  $m$ , which we denote as follows.

$$\vec{e} \sim m(\theta) \tag{15}$$

Note that we overload  $m$  to describe both the CMDP tuple and the probability distribution the CMDP yields when being executed with a policy and constraints.

A policy that optimizes the constrained optimization problem (c.f. Equation 13) can be synthesized with safe reinforcement learning [31]. One approach is to formulate the problem as a Lagrange function and use it as a reward function for a reinforcement learning algorithm [4]. Given an episode sampled from an MDP, we can compute cumulative return  $\mathcal{R}$  and cost  $\mathcal{C}_\varphi$  for a given episode and a given path formula  $\varphi$  as defined in Section 2.2. In general, we can transform the problem

$$\max \mathcal{R} \text{ s.t. } \mathcal{C}_\varphi = 0 \tag{16}$$

to its Lagrange formulation

$$\max \mathcal{R} - \lambda \mathcal{C}_\varphi \tag{17}$$

where  $\lambda \in \mathbb{R}^+$  is a Lagrangian multiplier [13]. Without loss of generality, we use an alternative formulation of Eq. 17 where  $\lambda \in (0, 1)$ .

$$\max \mathcal{R} - (1 - \lambda)\mathcal{C}_\varphi \tag{18}$$

We outline the general process of safe RL with function approximation in Algorithm 1. Note that the expectation is not given in line 7 due to sampling of episodes, returns and costs. Also note that Algorithm 1 does not ensure safety while learning, but only when converging to a solution of the Lagrangian. The key challenge in this approach is to determine an appropriate  $\lambda$  and to adjust it effectively over the learning process.

---

**Algorithm 1** Safe RL

---

- 1: **procedure** SAFE RL(CMDP  $m \in M$ , path formula  $\varphi$ )
  - 2:   initialize parameters  $\theta$
  - 3:   **while** learning **do**
  - 4:     generate episodes by sampling from  $m$
  - 5:     determine return  $\mathcal{R}$  and cumulative cost  $\mathcal{C}_\varphi$
  - 6:     determine  $\lambda$
  - 7:     update  $\theta$  wrt.  $\max_\theta \lambda \mathcal{R} - (1 - \lambda)\mathcal{C}_\varphi$
-

## 2.4 Verification: Bayesian Model Checking Algorithm V

The empirical nature of reinforcement learning necessitates quantification of confidence about properties of learned policies. We resort to statistical model checking [47] to quantify confidence and verify policies accordingly, in particular to Bayesian model checking (BMC) [41, 65, 15]. Bayesian model checking models epistemic uncertainty about satisfaction probability via sequential Bayesian update of the posterior distribution. We leverage BMC in two ways: To guide the learning process towards feasible solutions, and to verify synthesized policies.

Executing a policy in  $m$  generating an episode either satisfies or violates a given bounded path formula without probabilistic operator. Thus, we can treat the generation of multiple episodes as Bernoulli experiment with a satisfaction probability  $p_{\text{sat}}$ . We want to estimate this probability in order to check whether it complies with our probabilistic constraint  $p_{\text{req}}$ , that is  $p_{\text{sat}} \geq p_{\text{req}}$ .

Rather than doing a point estimate of  $p_{\text{sat}}$  via maximum likelihood estimation we assign a plausibility to each possible  $p_{\text{sat}} \in (0, 1)$ , yielding a Bayesian estimate. We assign a prior distribution to all possible values of  $p_{\text{sat}}$ , and then compute the posterior distribution based on the observations  $O$  of cost constraint satisfaction or violation. The posterior distribution allows us to quantify our confidence in whether a given required satisfaction probability is met.

In general, the posterior is proportional to the prior  $P(p)$  and the likelihood of the observations given this prior.

$$P(p|O) \propto P(O|p)P(p) \quad (19)$$

In the particular case of a Bernoulli variable, the conjugate prior is the Beta distribution [25], meaning that prior and posterior distribution are of the same family (the Beta distribution in our case). The Beta distribution is defined by two parameters  $\alpha, \beta \in \mathbb{N}^+$ , which are given by the observed count of positive and negative results of the Bernoulli experiment. We use a uniform prior  $\alpha, \beta = 1$  over possible values of  $p_{\text{sat}}$ , assigning the same plausibility to all possible values before observing any data. This yields the following equality when assuming  $s$  satisfactions and  $v$  violations of a given constraint.

$$P(p_{\text{sat}}|s, v) = \text{Beta}(s + 1, v + 1) \quad (20)$$

This update yields a posterior distribution over the possible values of  $p_{\text{sat}}$  given the observation of satisfaction and violation. We can now compute the probability mass  $c_{\text{sat}}$  of this posterior that lies above the required probability  $p_{\text{req}}$  to obtain a confidence about the current system satisfying our probabilistic constraint.

$$c_{\text{sat}} = \int_{p_{\text{req}}}^1 P(p_{\text{sat}}) dp_{\text{sat}} = 1 - \text{Beta}(s + 1, v + 1).\text{cdf}(p_{\text{req}}) \quad (21)$$

Here, cdf denotes the cumulative density function of the Beta distribution.

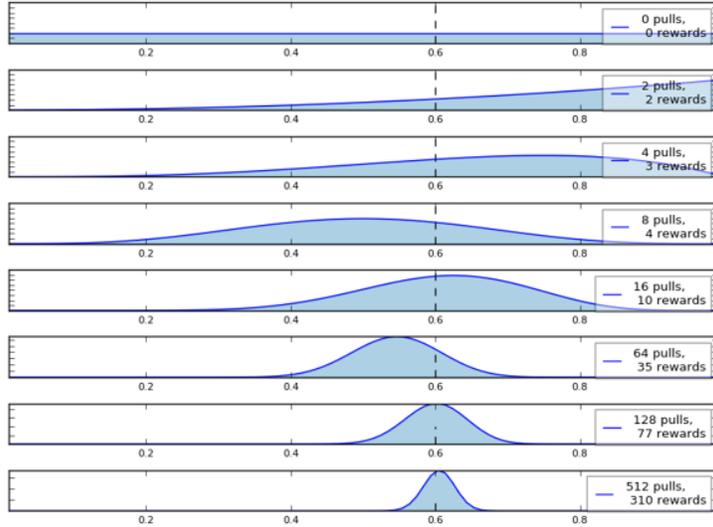


Figure 1: Exemplary evolution of the belief distribution wrt.  $\mu$  for a Bernoulli bandit. See text for details.

Figure 1 shows the evolution of the belief distribution for an example Bernoulli bandit with a probability  $\mu = 0.6$  of providing a reward when pulling the bandit. We want to infer  $\mu$  by sampling from the bandit, where each pull either yields a reward or no reward. Initially, no data is observed and the prior is uniform, representing that all values for  $\mu$  are equally likely a priori (before seeing any data). The images show the evolution of the belief distribution for an increasing number of pulls and correspondingly observed rewards from top to bottom. An increasing number of pulls yields a sharper (i.e. less uncertain) estimate of  $\mu$ . The probability mass (i.e. the blue integral) to the right of any value for  $\mu$  is the confidence that the true  $\mu$  lies above the value. For example, the integral to the right of 0.6 denotes the confidence that  $\mu$  is indeed larger than 0.6.

We can think of the environment in combination with a particular policy as our Bernoulli bandit, providing a reward when given constraints are satisfied. The agent wants to infer the satisfaction probability, and can quantify its confidence in the result using Bayesian modeling as outlined above.

Algorithm 2 shows the pseudo code for Bayesian verification (BV) of a policy that is parametrized with parameters  $\theta$ . It repeatedly generates an episode (line 4) and computes its cumulative cost (line 5), evaluates whether it satisfies the safety requirement by testing whether the episode cost equals zero (line 6), and updates its confidence in satisfaction accordingly (line 7). BV terminates when the confidence reaches a given bound (lines 8 and 9).

---

**Algorithm 2** Bayesian verification

---

```
1: procedure BV(Policy parameters  $\theta$ , CMDP  $m \in M$ , requirement  
    $\mathbb{P}_{\geq p_{\text{req}}}(\varphi)$  and  $\mathbb{C}_{\geq c_{\text{req}}}$ )  
2:    $s, v \leftarrow 0$   
3:   loop  
4:      $\vec{e} \sim m(\theta)$   
5:     compute  $\mathcal{C}_\varphi$  wrt.  $\vec{e}$   
6:     update  $s$  or  $v$  wrt.  $\mathcal{C}_\varphi =? 0$   
7:     determine  $c_{\text{sat}}$  wrt. Eq. 21  
8:     if  $c_{\text{sat}} \geq c_{\text{req}}$  then return true  
9:     if  $1 - c_{\text{sat}} \geq c_{\text{req}}$  then return false
```

---

While this approach allows to verify a given policy after training, it does not directly provide a way to synthesize a policy that is likely to be verified. We will provide an approach to this problem in the next section.

### 3 Safe Neural Evolutionary Strategies

The previous section outlined a methodology for engineering safe policy synthesis based on specification as a CMDP, safe RL and Bayesian verification. In this section, we propose Safe Neural Evolutionary Strategies (SNES) for learning policies that are likely to be positively verified. SNES weights return and cost in the process of policy synthesis based on Bayesian confidence estimates obtained in the course of learning.

#### 3.1 Evolutionary Strategies

Evolutionary strategies (ES) [59] is a gradient free, search-based optimization algorithm that has shown competitive performance in reinforcement learning tasks using deep learning for function approximation. ES is attractive as it is not based on backpropagation and can therefore be parallelized straightforwardly. Also, it does not require expensive GPU hardware for efficient computation.

The basic ES procedure is shown in Algorithm 3. ES works by maintaining the parameters  $\theta$  of the current solution. It then generates  $N \in \mathbb{N}^+$  slightly perturbed offspring from this solution to be evaluated on the optimization task  $f$ , for example optimizing expected episode return of a policy in an MDP (lines 4 to 7). In our case, we use  $N$  normally distributed samples with a mean of 0 and a standard deviation of  $\sigma$ . The current solution is then updated by moving the solution parameters into the direction of offspring weighted by their respective return, in expectation increasing effectiveness of the solution (lines 8 and 9).

We normalize a set of values  $X \in \mathbb{R}^{\mathbb{N}^+}$  to zero mean and unit standard deviation with the following normalization procedure.

$$\text{normalize}(X) =_{\text{def}} \forall x \in X : x \leftarrow \frac{x - \text{mean}(X)}{\text{std}(X)} \quad (22)$$

---

**Algorithm 3** ES

---

```
1: procedure ES(population size  $N \in \mathbb{N}^+$ , perturbation rate  $\sigma \in \mathbb{R}^+$ , learning
   rate  $\alpha \in (0, 1]$ , task  $f : \Theta \rightarrow \mathbb{R}$ )
2:   initialize parameters  $\theta$ 
3:   while learning do
4:     for  $i \in \{1, \dots, N\}$  do
5:        $\epsilon_i \sim \text{Normal}(0, \sigma, N)$  ▷ perturb offspring
6:        $\theta_i \leftarrow \theta + \epsilon_i$ 
7:        $\mathcal{R}_i \leftarrow f(\theta_i)$  ▷ determine return
8:       normalize( $\bigcup_i \mathcal{R}_i$ ) ▷ update solution
9:      $\theta \leftarrow \theta + \frac{\alpha}{\sigma N} \sum_{j=1}^N \mathcal{R}_j * \epsilon_j$ 
```

---

### 3.2 Safe Neural Evolutionary Strategies

Safe Neural Evolutionary Strategies (SNES) combines ES with Bayesian verification in the learning process to adaptively weight return and cost in the course of policy synthesis such that the resulting policy is likely to be positively verified.

SNES is an instance of safe RL based on basic ES for parameter optimization, additionally performs a Bayesian verification step in each iteration and uses the current verification confidence for balancing return maximization and constraint satisfaction. The resulting confidence estimate in positive verification is then used to determine the weighting  $\lambda$  of return and cost.

In order to account for confidence requirements, the confidence estimate  $c_{\text{sat}}$  from Bayesian verification is set in relation to the required confidence  $c_{\text{req}}$  such that if confidence in constraint satisfaction is lower than required, only costs are reduced in the parameter update. If the constraint is satisfied with enough confidence, the influence of return is gradually increased.

$$\lambda \leftarrow \frac{\max(0, c_{\text{sat}} - c_{\text{req}})}{1 - c_{\text{req}}} \quad (23)$$

SNES is shown in Algorithm 4. SNES generates offspring by adding noise to the parameters of the current policy (lines 7 and 8). It evaluates the offspring by generating an episode from the MDP (line 9) and computes its return and cost (line 10), and whether it satisfies or violates the requirement (line 11). It uses this information to update the Lagrangian of the constrained optimization problem to weight return and cost in the learning process (lines 12 to 14). Learning is done by updating parameters of a policy, weighted accordingly to normalized return and cost (lines 15 to 18).

---

**Algorithm 4** Safe Neural Evolutionary Strategies (SNES) for policy synthesis under probabilistic constraints

---

```

1: procedure SNES(population size  $N \in \mathbb{N}^+$ , perturbation rate  $\sigma \in \mathbb{R}^+$ ,
   learning rate  $\alpha \in (0, 1]$ , CMDP  $m \in M$ , requirement  $\mathbb{P}_{\geq p_{\text{req}}}(\varphi)$  and  $\mathbb{C}_{\geq c_{\text{req}}}$ )
2:   initialize parameters  $\theta$ 
3:    $s, v \leftarrow 0$  ▷ initially no satisfying or violating episodes
4:    $\lambda \leftarrow 1$ 
5:   while learning do
6:     for  $i \in \{1, \dots, N\}$  do
7:        $\epsilon_i \sim \text{Normal}(0, \sigma, N)$  ▷ perturb current parameters
8:        $\theta_i \leftarrow \theta + \epsilon_i$ 
9:        $\vec{e} \sim m(\theta_i)$  ▷ evaluate perturbation
10:      compute  $\mathcal{R}_i, \mathcal{C}_{\varphi, i}$  wrt.  $\vec{e}$ 
11:       $s \leftarrow s + \mathbb{I}(\mathcal{C}_{\varphi, i} = 0)$  ▷ count satisfying episodes
12:       $v \leftarrow v + N - s$  ▷ number of violating episodes
13:       $c_{\text{sat}} \leftarrow 1 - \text{Beta}(s + 1, v + 1).\text{cdf}(p_{\text{req}})$  ▷ determine confidence
14:       $\lambda \leftarrow \frac{\max(0, c_{\text{sat}} - c_{\text{req}})}{1 - c_{\text{req}}}$  ▷ set  $\lambda$  wrt. requirement
15:      normalize( $\bigcup_i \mathcal{R}_i$ ) ▷ normalize return and cost
16:      normalize( $\bigcup_i \mathcal{C}_{\varphi, i}$ )
17:       $\theta \leftarrow \theta + \frac{\alpha \lambda}{\sigma N} \sum_{j=0}^N \mathcal{R}_j * \epsilon_j$  ▷ update parameters
18:       $\theta \leftarrow \theta - \frac{\alpha(1-\lambda)}{\sigma N} \sum_{j=0}^N \mathcal{C}_{\varphi, j} * \epsilon_j$ 

```

---

**Remark: SNES vs. maximum likelihood calibration of the Log-likelihood** To show the effect of the Bayesian treatment and the necessity of computing confidence for adapting  $\lambda$  in the learning process, we compared SNES to a naive variant for tuning  $\lambda$  (for results cf. Section 4.1.2). Here, we use a maximum likelihood estimate  $\hat{p}_{\text{sat}}$  for satisfaction probability and adjust  $\lambda$  according to the following rule, replacing lines 13 and 14 in Algorithm 4.

$$\hat{p}_{\text{sat}} \leftarrow \frac{s}{N} \tag{24}$$

$$\lambda \leftarrow \frac{\max(0, \hat{p}_{\text{sat}} - p_{\text{req}})}{1 - p_{\text{req}}} \tag{25}$$

Note that the naive approach does not account for confidence in its result.

## 4 Experiments

In this section, we report our empirical results obtained when evaluating SNES for two domains:<sup>1</sup>

---

<sup>1</sup>Code for our experiments is available at <https://github.com/lenzbelzner/psyco>.

- The *Particle Dance* domain has continuous state and action spaces, and a fixed episode length. Its initial policy is likely to satisfy given constraints when starting the learning process.
- The *Obstacle Run* domain has discrete state and actions spaces, and has an adaptive episode length based on a termination criterion. Here, the initial policies are likely to violate given constraints.

For each of the domains, we describe the setup of our experiment by presenting the respective domain, the rewards, costs, and requirements, the neural network for modeling the policy, and the parameters for performing the experiments. Then we analyze the episode return, satisfaction probability and confidence, and show the results of Bayesian verification for both domains. For the *Particle Dance* domain, we additionally compare the SNES calibration of the Langrangian with a maximum likelihood approach.

## 4.1 Particle Dance

In the *Particle Dance* domain, an agent has to learn to follow a randomly moving particle as closely as possible while keeping a safe distance. The *Particle Dance* domain is continuous with fixed episode length, and an initial policy is likely to satisfy given constraints.

### 4.1.1 Setup

Agent and particle have a position  $x \in [-2, 2]^2$  and a velocity  $v \in [-0.1, 0.1]^2$ . The state space  $S$  describes the positions and velocities of both agent and particle. We restrict positions and velocities to their respective boundaries by clipping any exceeding values. The state also keeps count of agent-particle collisions  $n_c \in \mathbb{N}$  (see below for their definition). Note that the collision counter is not observed by the agent in our setup, i.e. is not used as input for training and querying its policy.

$$S : [-2, 2]^4 \times [-0.1, 0.1]^4 \times \mathbb{N} \quad (26)$$

The initial positions are sampled from  $[-1, 1]^4$  uniformly at random. The initial velocities are fixed to zero, as is the initial collision counter.

$$\rho : \mathcal{U}([-1, 1]^4) \times [0, 0]^4 \times 0 \quad (27)$$

The agent can choose its acceleration at each time step. This yields the continuous action space  $A$ .

$$A : [-0.1, 0.1]^2 \quad (28)$$

The agent is accelerated at each time step by a value  $a \in [-0.1, 0.1]^2$ . The particles' acceleration is sampled uniformly at random at each time step. Positions are updated wrt. current velocities. We define a collision radius  $d_{\min} \in \mathbb{R}^+$  and induce a cost when the agent is closer to the particle than this radius and

update  $n_c$  accordingly. Let  $s \in S$  be the systems current state and  $a \in A$  be the action executed by the agent, then the transition distribution  $T : p(S|S, A)$  is given by the following sequence of assignments:

$$s = (x_{\text{agent}}, x_{\text{particle}}, v_{\text{agent}}, v_{\text{particle}}, n_c)$$

$$T(s, a) \sim \begin{cases} v_{\text{particle}} \leftarrow v_{\text{particle}} + \mathcal{U}[-0.1, 0.1]^2 \\ x_{\text{particle}} \leftarrow x_{\text{particle}} + v_{\text{particle}} \\ v_{\text{agent}} \leftarrow v_{\text{agent}} + a \\ x_{\text{agent}} \leftarrow x_{\text{agent}} + v_{\text{agent}} \\ n_c \leftarrow n_c + \mathbb{I}(d(x_{\text{agent}}, x_{\text{particle}}) < d_{\min}) \end{cases}$$

Note that  $T$  is not known by the agent and detailed here only to render our experimental setup reproducible.

**Reward and Safety Predicate** The agent gets a reward at each step of an episode motivating it to get as close to the particle as possible. We define a collision radius  $d_{\min} \in \mathbb{R}^+$  and induce a cost when the agent is closer to the particle than this radius. We also define a collision counter  $n_c \in \mathbb{N}$  as an atomic proposition of states which allows us to specify a number of collisions  $n_{\max}$ . Note that  $n_c$  is not observed by the agent in our experiments. We set  $n_{\max} \in 1, 4$  and  $d_{\min} = 0.1$  in our experiments. Let  $s' = (x'_{\text{agent}}, x'_{\text{particle}}, v'_{\text{agent}}, v'_{\text{particle}}, n'_c)$ .

$$R(s, a, s') = -d(x'_{\text{agent}}, x'_{\text{particle}}) \quad (29)$$

$$\phi = d(x'_{\text{agent}}, x'_{\text{particle}}) \geq d_{\min} \vee n'_c \leq n_{\max} \quad (30)$$

**Requirements and Cost** The reward computes the negative distance between particle and agent. Minimizing the distance means maximizing the reward. Thus the optimizing goal is to maximize the expectation of the return  $\mathcal{R}$  (see (5)):

$$\mathbf{Goal} \text{ Optimize } \textit{Return} : \max \mathbb{E}(\mathcal{R}) \quad (31)$$

The safety constraint requires the agent to keep a minimum distance of the particle except in  $n_{\max}$  cases. We set the required probability for satisfying the constraint  $p_{\text{req}} = 0.85$  and the required confidence  $c_{\text{req}} = 0.98$ .

$$\mathbf{Goal} \text{ Constraint } \textit{BoundedCollisions} : \mathbb{P}_{\geq 0.85}(\square\phi) \text{ and } \mathbb{C}_{\geq 0.98} \quad (32)$$

The cost induced by the safety constraint is given by equation 10.

**Policy Network** We model the policy of our agents as a feedforward neural network with parameters  $\theta$ . Our network consists of an input layer with dimension 8 (position and velocity of agent and particle), a hidden layer with dimension 32, and has an output dimension of 2 (two dimensions of acceleration). Let  $\theta = \{\theta_1, \theta_2\}$  be the networks' weights in the input and hidden layer respectively, and let  $f_1$  and  $f_2$  be non-linear activation functions, with  $f_1$  being

a rectified linear unit [33] and  $f_2$  being  $\tanh$  in our case. Then, the networks output is given as follows.

$$y \leftarrow f_2(\theta_2 f_1(\theta_1 x + 1) + 1) \quad (33)$$

**Other Parameters** We report our results for episode length  $n = 50$ , population size  $N = 20$ , learning rate  $\alpha = .01$  and perturbation rate  $\sigma = .1$ . Experiments with other parameters yielded similar results.

**Experimental Setup** Each experimental run comprised learning a policy with SNES over 60000 episodes. Every 1000 episodes, we performed Bayesian verification for a maximum of 1000 episodes (outside the learning loop of SNES) to evaluate the policy synthesized by SNES up to that point.

We repeated the experiment five times and show mean values as solid lines and standard deviation by shaded areas in our figures.

**A Note on Performance** The performance of SNES depends on the neural network architecture, the simulation of the MDP transitions and the used hardware. In our setup using a laptop computer, each step took less than a millisecond of execution time.

#### 4.1.2 Results

The SNES agent learns to follow the particle closely. In Figure 2 we see sample trajectories of the particle and the agent (color gradients denote time). We observe that the synthesized policy learned the task to follow the particle successfully.

We can observe the effect of SNES learning unconstrained and constrained policies on the obtained episode return in Figure 3. Return and constraint define a Pareto front in our domain: Strengthening the constraint reduces the space of feasible policies, and also reduces the optimal return that is achievable by a policy due to increased necessary caution when optimizing the goal, i.e. when learning to follow the particle as close as possible.

Figure 4 shows the proportion of episodes that satisfy the given requirement. We can see that the proportion closely reaches the defined bound, shown by the dashed vertical line. Note that the satisfying proportion is closely above the required bound.

Figure 5 shows the confidence of the learning agent in its ability to satisfy the given requirement based on the observations made in the learning process so far. The confidence is determined from the Beta distribution maintained by SNES over the course of training. Note that the confidence is mostly kept above the confidence requirement given in the specification. This shows SNES is effectively incorporating observations, confidence and requirement into its learning process.

Figure 6 shows the results of Bayesian verification (see Alg. 2) performed throughout the learning process every 1000 episodes. Here, we fix the current

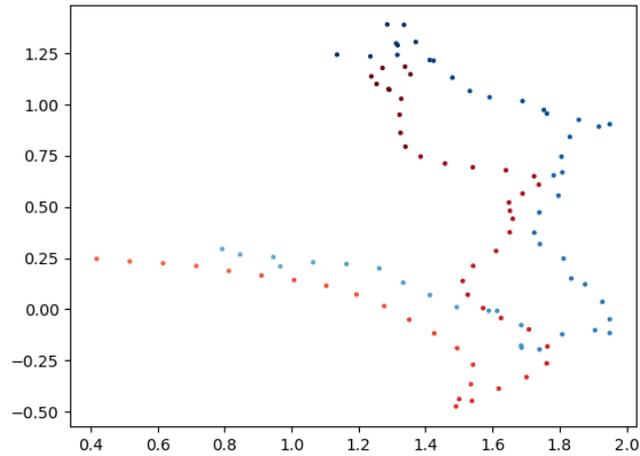


Figure 2: Sample trajectories of the particle (light to dark blue, color gradient denotes time) and the agent (light to dark red). The shown trajectory has been learned by an unconstrained agent and achieves a return of ca.  $-15$  without incurring a collision.

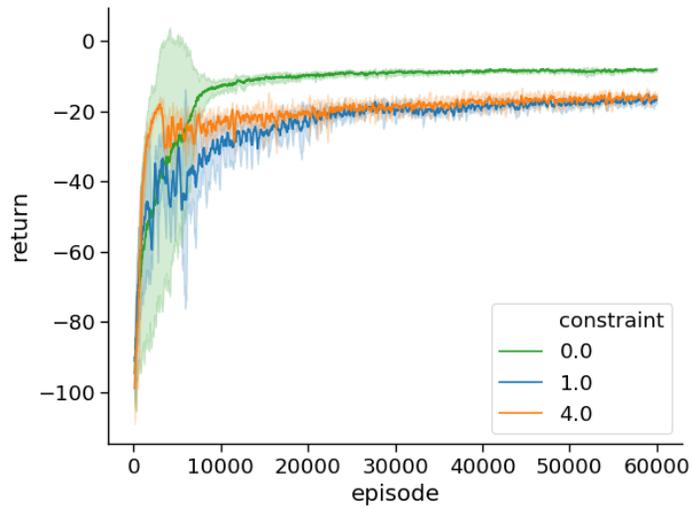


Figure 3: Episode return for various constraints. Constraint 0.0 denotes unconstrained policy synthesis.

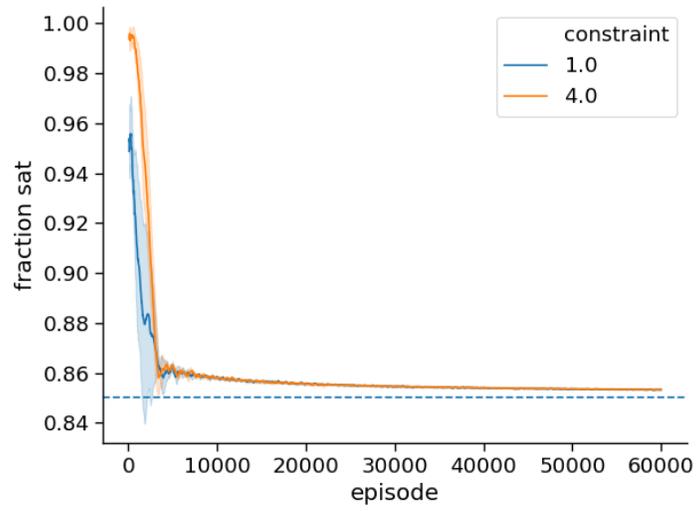


Figure 4: Proportion of episodes satisfying cost requirement.

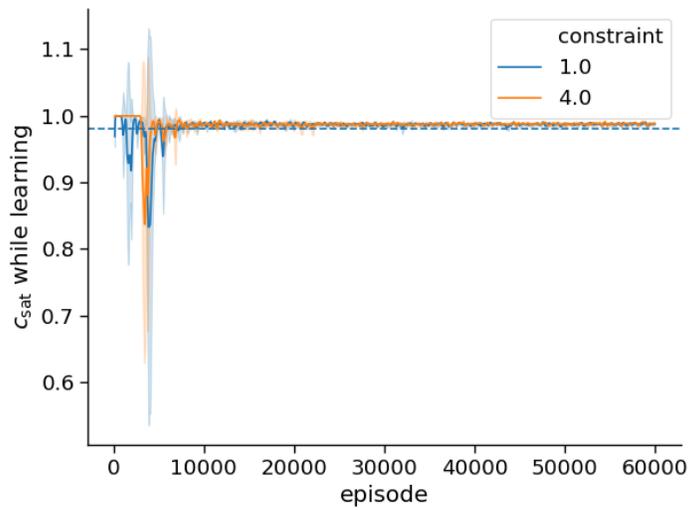


Figure 5: Confidence  $c_{\text{sat}}$  in satisfying specification based on observations in the course of learning.

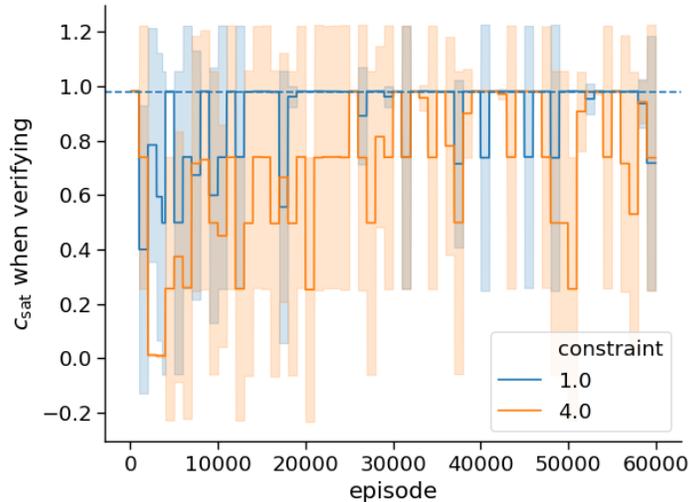


Figure 6: Confidence obtained when exhaustively verifying stationary current policies over the course of learning with Bayesian verification BV every 1000 episodes.

policy, and perform Bayesian verification of the policy wrt. the given requirement. The quantity measured is the confidence in requirement satisfaction after either surpassing  $c_{\text{req}}$  (i.e. the policy satisfies the requirement with high confidence), falling below  $1 - c_{\text{req}}$  (i.e. the policy violates the requirement with high confidence) or after a maximum of 1000 verification episodes. We can see that the confidence in having learned a policy that satisfies the requirement is increasing over the course of training. However, as the goals of optimizing return and satisfying collision constraints are contradictory in the Particle Dance domain, SNES may still produce policies that eventually violate the specification.

Figures 7, 8 and 9 show return and collisions (i.e. cost) obtained, split by episodes that satisfy the constraint and those that violate it. We can see the violating episodes are more effective in terms of return but keep collisions well below the requirement, highlighting again the Pareto front of return and cost given by our domain. Note that we smooth the shown quantity over the last 100 episodes, and that collision only can take discrete values. This may explain that the shown quantity is well below the theoretically given boundaries (one or four in our case). SNES is able to learn policies that exploit return in the defined proportion of episodes, and to optimize wrt. the Pareto front of return and cost otherwise.

**Results on SNES vs. maximum likelihood calibration of the Lagrangian** We compared SNES’ Bayesian approach to calibrating  $\lambda$  (Eq. 23) to

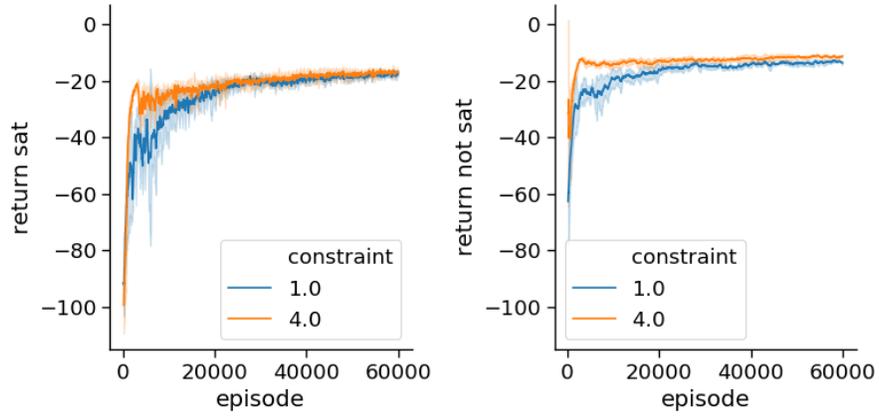


Figure 7: Return of episodes satisfying (left) and violating (right) constraints.

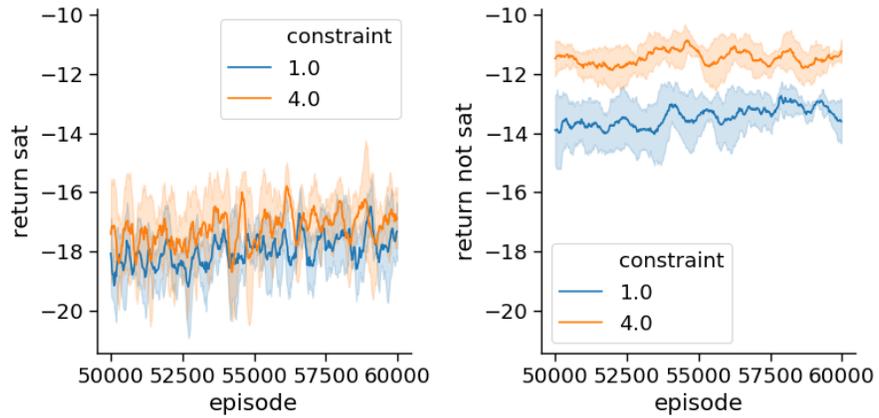


Figure 8: Return of episodes satisfying (left) and violating (right) constraints as shown in Figure 7, showing the final episodes in more detail. The less constrained policy is able to optimize its return more effectively.

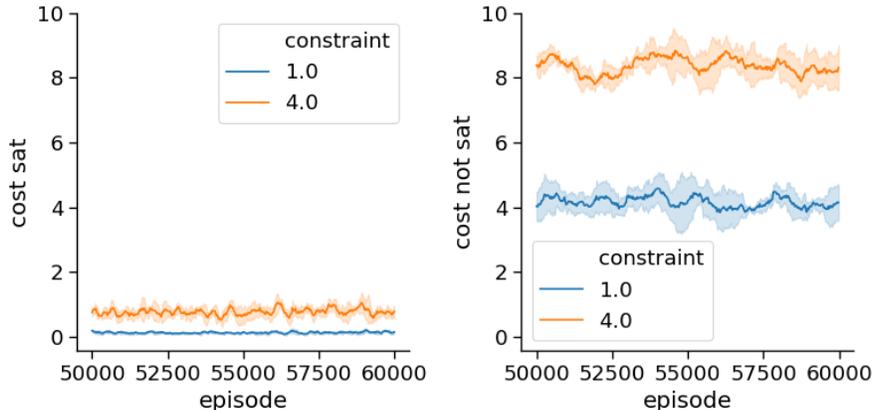


Figure 9: Number of collision events (i.e. cost) of episodes satisfying (left) vs. violating (right) constraints.

a maximum likelihood variant (Eq. 25). Figures 10, 11 and 12 show return and cost for both approaches, separated by satisfying and violating episodes. We show the results for  $n_{\max} = 1$ , aggregated for 12 repetitions of the experiment.

The MLE approach over-satisfies the given constraint, and consequently does not yield as high returns as SNES. In contrast, SNES is able to find a local Pareto optimum of return and constraint satisfaction.

## 4.2 Obstacle Run

In the *Obstacle Run* domain, an agent has to reach a target position while not colliding with a moving obstacle. In contrast to *Particle Dance*, *Obstacle Run* has discrete states and actions, and episodes terminate when the agent reaches the target position. Also, an initial policy is likely to violate the given constraints when starting the learning process.

### 4.2.1 Setup

The setup is similar to the Particle Dance, except that positions are discrete with  $x \in \{0, \dots, 4\}^2$  and that the agent does not change the velocity but only the movement direction in each step. The state space  $S : \{0, \dots, 4\}^4 \times \mathbb{N}$  consists of the positions of agent and obstacle and of a collision counter  $n_c \in \mathbb{N}$ . As in Particle Dance, the collision counter is not used as input for training and querying the agent’s policy and positions are restricted to their respective boundaries by clipping any exceeding values.

The initial positions are sampled from  $\{0, \dots, 4\}^4$  uniformly at random. The initial collision counter is set to zero.

$$\rho : \mathcal{U}(\{0, \dots, 4\}^4) \times 0 \tag{34}$$

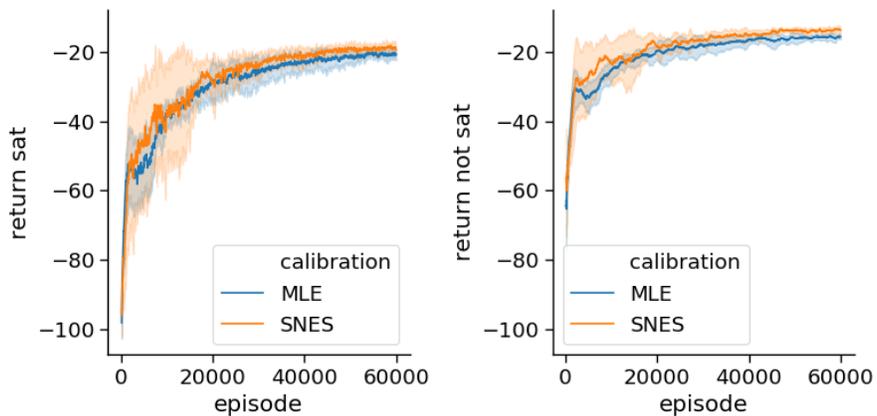


Figure 10: Return of episodes satisfying (left) and violating (right) constraints for SNES and MLE of  $\hat{p}_{\text{sat}}$ .

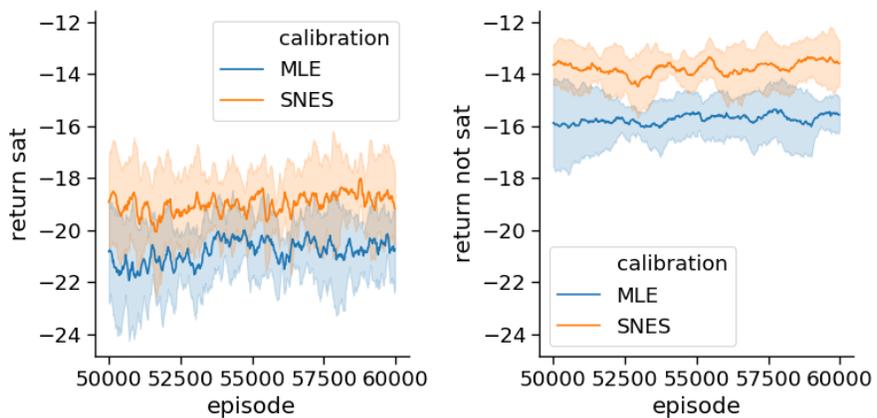


Figure 11: Return of episodes satisfying (left) and violating (right) constraints for SNES and MLE of  $\hat{p}_{\text{sat}}$  as shown in Figure 10, showing the final episodes in more detail. SNES is able to exploit the Pareto front of optimization and constraints more effectively.

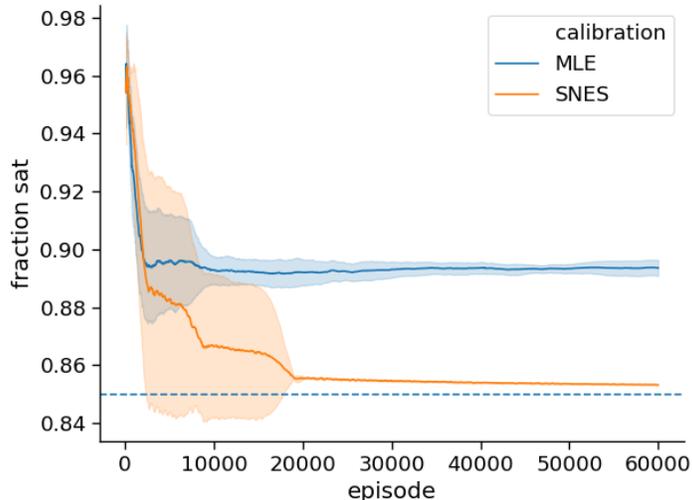


Figure 12: Proportion of satisfying episodes for SNES and MLE of  $\hat{p}_{\text{sat}}$ .

The agent can choose its movement direction at each time step. This yields the discrete action space  $A$ .

$$A : \{(0, 0), (1, 0), (0, 1), (-1, 0), (0, -1)\} \quad (35)$$

The agent moves at each time step by according to its chosen action. The obstacle's movement is sampled uniformly at random from  $A$  at each time step. Positions are updated wrt. actions. A collision occurs if agent and obstacle share the same position, and update  $n_c$  accordingly. Let  $s \in S$  be the systems current state and  $a \in A$  be the action executed by the agent, then the transition distribution  $T : p(S|S, A)$  is given by:

$$s = (x_{\text{agent}}, x_{\text{obstacle}}, n_c)$$

$$T(s, a) \sim \begin{cases} x_{\text{obstacle}} \leftarrow x_{\text{obstacle}} + \mathcal{U}(A) \\ x_{\text{agent}} \leftarrow x_{\text{agent}} + a \\ n_c \leftarrow n_c + \mathbb{I}(x_{\text{agent}} = x_{\text{obstacle}}) \end{cases}$$

Note that  $T$  is not known by the agent and detailed here only to render our experimental setup reproducible.

We fix the target position  $x_{\text{target}} = (0, 0)$  and end an episode if  $x_{\text{agent}} = x_{\text{target}}$ .

**Requirements, Reward, and Cost** The agent gets a reward of  $-1$  at each step of an episode motivating it to reach the target as fast as possible. The

optimizing goal is to maximize the expectation of the return  $\mathcal{R}$  (see (5)). Note that this is achieved by reaching the target position as fast as possible.

$$R(s, a, s') = -1 \quad (36)$$

$$\mathbf{Goal\ Optimize\ Return} : \max \mathbb{E}(\mathcal{R}) \quad (37)$$

The safety constraint requires the agent to avoid the (moving) obstacle except in  $n_{max}$  cases. We set the required probability for satisfying the constraint  $p_{req} = 0.9$  and the required confidence  $c_{req} = 0.98$ .

$$\phi = x_{agent} \neq x_{target} \vee n_c \leq n_{max} \quad (38)$$

$$\mathbf{Goal\ Constraint\ BoundedCatches} : \mathbb{P}_{\geq 0.9}(\Box\phi) \text{ and } \mathbb{C}_{\geq 0.98} \quad (39)$$

The cumulative cost is again given by equation (10). In our experiments we set  $n_{max} \in \{1, 4\}$ .

**Policy Network** As in Particle Dance, the policy network is a feedforward neural network with a hidden layer of dimension 32 and its output is computed by equation (33). The input layer has dimension 4 (position of agent and obstacle) and the output layer has dimension 5.

Actions are chosen by identifying the index of the maximum output and choosing one of the five available actions accordingly.

**Other Parameters** As for Particle Dance we report our results for a maximum episode length  $n = 50$ , population size  $N = 20$ , learning rate  $\alpha = .01$  and perturbation rate  $\sigma = .1$ . Experiments with other parameters yielded similar results.

**Experimental Setup** Each experimental run comprised learning a policy with SNES over 20000 episodes. Every 1000 episodes, we performed Bayesian verification for a maximum of 1000 episodes (outside the learning loop of SNES) to evaluate the policy synthesized by SNES up to that point.

We repeated the experiment five times and show mean values as solid lines and standard deviation by shaded areas in our figures.

#### 4.2.2 Results

We can observe the effect of SNES learning unconstrained and constrained policies on the obtained episode return in Figure 13. Constraint 0.0 denotes unconstrained policy synthesis. Return and constraint define a Pareto front in our domain: Strengthening the constraint reduces the space of feasible policies, and also reduces the optimal return that is achievable by a policy due to increased necessary caution when optimizing the goal, i.e. when learning to avoid the obstacle as good as possible.

Figure 14 shows the proportion of episodes that satisfy the given requirement. We can see that the easily surpasses defined bound, shown by the dashed

vertical line. In contrast to *Particle Dance*, the agent is able to find policies that easily satisfy the given constraint by reaching the target as fast as possible, thus reducing the probability of a collision.

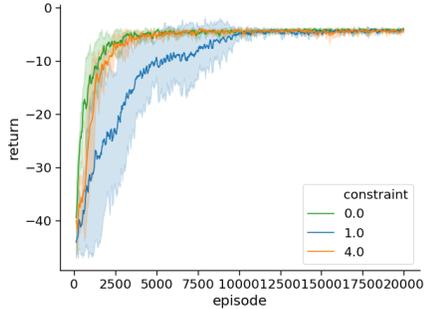


Figure 13: *Obstacle Run*: Episode return for various constraints.

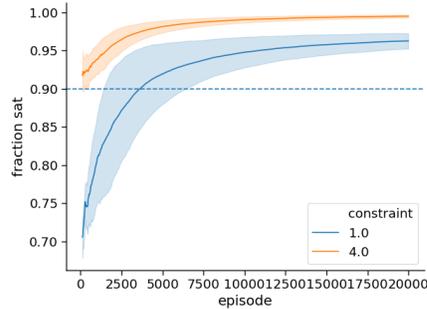


Figure 14: *Obstacle Run*: Proportion of episodes satisfying cost requirement.

Figure 15 shows the confidence of the learning agent in its ability to satisfy the given requirement based on the observations made in the learning process so far. The confidence is determined from the Beta distribution maintained by SNES over the course of training. The agent is able to learn a policy that allows to both optimize return and satisfy the given constraint in a stable way.

Figure 16 shows the results of Bayesian verification performed throughout the learning process every 1000 episodes. In contrast to the *Particle Dance* domain, in the *Obstacle Run* domain the agent is likely to violate the constraint at the start of the learning process. Therefore, in the beginning it tends to underestimate its ability to satisfy the constraint, as can be seen from comparing its confidence while learning (Figure 15) with the (offline) confidence acquired from verifying (Figure 16).

Figures 17 and 18 show return and collisions (i.e. cost), split by episodes that satisfy the constraint and those that violate it. We can see the violating episodes are more effective in terms of return but keep collisions well below the requirement, highlighting again the Pareto front of return and cost given by our domain. SNES is able to learn policies that exploit return in the defined proportion of episodes, and to optimize wrt. the Pareto front of return and cost otherwise.

## 5 Related Work

For synthesizing policies of autonomous and adaptive systems, PSyCo comprises a systematic development method, algorithms for safe and robust reinforcement learning, and a Bayesian verification method which is related to MDP model checking and statistical model checking approaches. In this section we discuss related work in these areas.

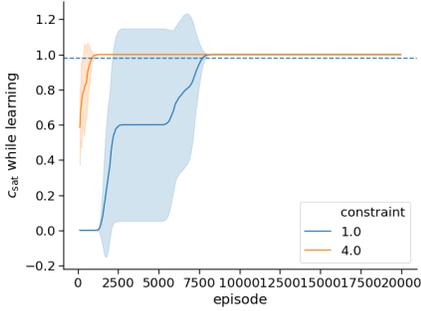


Figure 15: *Obstacle Run*: Confidence  $c_{\text{sat}}$  in satisfying specification based on observations in the course of learning.

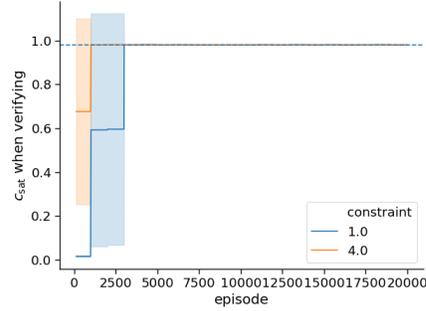


Figure 16: *Obstacle Run*: Confidence obtained when exhaustively verifying stationary current policies over the course of learning with Bayesian verification BV every 1000 episodes.

**Systematic Development of Adaptive Systems** PSyCo borrows its notion of goals from KAOS [24], an early method for goal-oriented requirements engineering. KAOS distinguishes hard and soft goals, is formally based on linear temporal logic, and proposes activities for refining the goals and deriving operation requirements which serve as the basis for system design. In contrast to PSyCo, it does neither cover system design nor implementation.

SOTA [1] is a modern requirements engineering method for autonomous and collective adaptive systems with a specific format for goals. Properties of goals can be analyzed by a modelchecking tool [2] based on LTL formulas and the LTSA modelchecker [49]. SOTA does neither address system design nor implementation; but it was used for requirements specification in the systematic construction process for autonomous ensembles [64] of the ASCENS project [63].

The ensemble development life cycle EDLC [18, 38] of ASCENS is a general agile development process covering all phases of system development and relating them with the “runtime feedback control loop” for awareness and adaptation. Its extension “Continuous Collaboration” [37] integrates a machine-learning approach into EDLC, but as EDLC, it does not address (policy) synthesis.

The following papers address more specific development aspects. In [16] a generic framework for modeling autonomous systems is presented which is centered around simulation-based online planning; Monte Carlo Tree Search and Cross Entropy Open Loop Planning are used for online generation of adaptive policies, but safety properties are not studied. In [26], Dragomir et al. propose an automated design process based on formal methods targeting partially observable timed systems. They describe how to automatically synthesize runtime monitors for fault detection, and recovery strategies for controller synthesis.

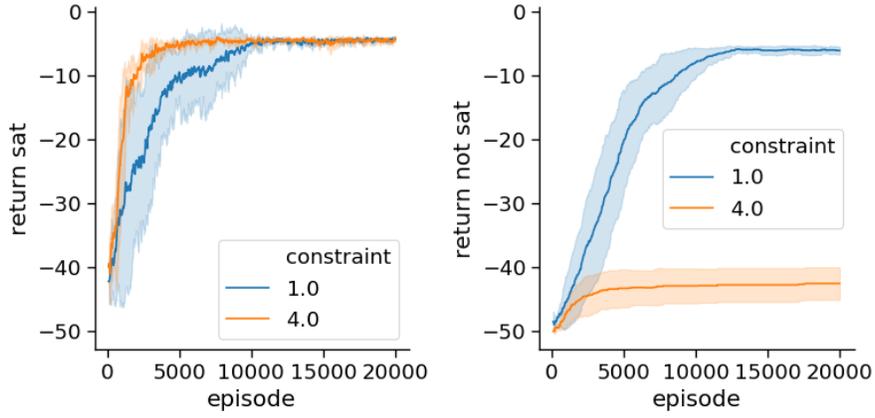


Figure 17: *Obstacle Run*: Return of episodes satisfying (left) and violating (right) constraints.

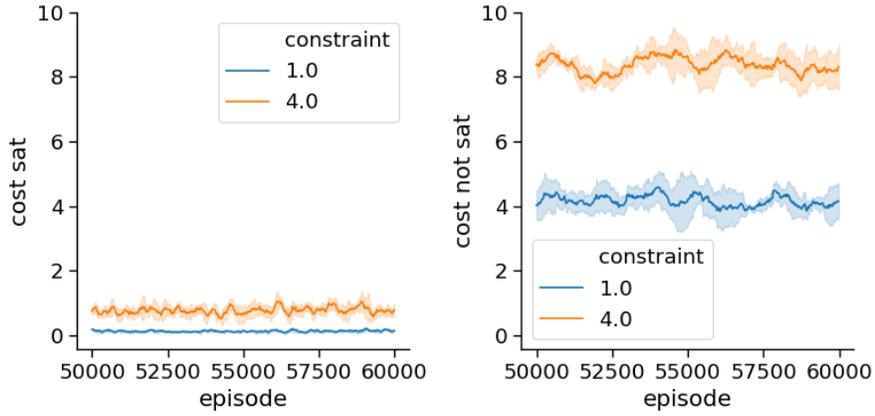


Figure 18: *Obstacle Run*: Number of collision events (i.e. cost) of episodes satisfying (left) vs. violating (right) constraints.

**Multi-Objective Reinforcement Learning and Preference-Based Planning**

Many applications have not only one but several optimising goals. Multi-objective optimisation [23] and multi-objective reinforcement learning [27] aim at simultaneously optimising several, usually conflicting objective functions. The solution is not unique, but consists of a set of so-called Pareto optimal policies which cannot be further improved in any objective without worsening at least another objective. Maintaining such multi-objective Pareto fronts is significantly more complex than dealing with a single optimizing goal. Thus in many approaches, the multi-objective problem is transformed into a single objective problem by using so-called scalarisation functions to combine the values of the different objectives into a single value. The resulting single objective problem can then be solved by standard learning and planning algorithms, for an overview see [57]. Using a Lagrangian approach, PSyCo transforms a multi-objective problem into a single objective problem.

Another related approach for optimizing behavior wrt. multiple user goals (i.e. preferences) is preference-based planning and learning where (user) preferences determine the quality of plans or a policies [12, 60]. Instead of using numerical rewards, policies and plans are compared w.r.t partial order “preference” relations.

**Model Checking for MDPs** PSyCo provides a framework for synthesizing policies that maximize return while being conform to a given probabilistic requirement specification. A related line of research is treating the problems of verifying general properties of a given MDP, such as reachability. Here, verification is done either for all possible policies, or for a particular fixed one turning the MDP into a Markov chain to be verified. [11, 9] present an overview of model checking techniques for qualitative and quantitative properties of MDPs expressed by LTL and PCTL formulas. For a recent review of this field see [10]. There are also software tools available, e.g. the PRISM model checker [45].

**Statistical Model Checking** Model checking is known to be time and memory consuming; its use is restricted to small and middle sized domains. Statistical model checking [47, 46] is a line of research for addressing this problem. In statistical model checking multiple executions of a system are observed and used for estimating the probabilities of system traces and giving results within confidence bounds. While our work builds on these ideas, policy synthesis is not a core aspect of statistical model checking: Usually information about the verification process is not induced into a learning process [47, 22, 45]. Other prior work has discussed the Bayesian approach to model checking based on the Beta distribution, which is a key component of the PSyCo framework. In contrast to our work, these works did not use information about the verification process to guide policy synthesis [41, 15].

**Safe and Robust Reinforcement Learning** Closely related to our approach of verifiable policy synthesis are works in the area of safe reinforcement

learning modeling the problem in terms of a constrained optimization problem [56, 32, 21, 28]. In contrast to our approach, these approaches do not reason about the statistical distribution of costs and corresponding constraint violation, nor do they provide a statistically grounded verification approach of given constraints.

[29, 30] propose a method for safe reinforcement learning which combines verified runtime monitoring with reinforcement learning. In contrast to our approach, their method requires a fully verified set of safe actions for a subset of the state space. While it is an interesting approach guaranteeing safety in the modeled subset, it is infeasible to perform exhaustive a-priori verification for very large or highly complex MDPs.

A notable exception is proposed in [20], which provides statistical optimization wrt. the cost distribution tail. In contrast, PSyCo provides (a) a framework integrating formal goal specifications and policy synthesis and (b) Bayesian verification of synthesized policies including statistical confidence in verification results. While the cost distribution tail adequately captures *aleatoric* uncertainty inherent to the domain, to the best of our knowledge, leveraging the Beta distribution to represent a learning agent’s *epistemic* confidence in constraint satisfaction and adapting the Lagrangian of a constraint optimization problem accordingly is a novel approach.

Other approaches to learning safe behavior are using a given or learned model of the environment [42, 34, 35, 8] or learning shields in addition to a policy assuring that only safe actions are executed [3, 17, 7, 43, 40]. Also, there are approaches to synthesizing policies maximizing the probability of satisfying LTL constraints without maximizing reward at the same time [36]. These approaches are orthogonal to our work, and using a learner’s confidence as a learning signal in these setups could be an interesting venue for further research.

Another direction to safe reinforcement learning is the use of adversarial methods, which treat the agent’s environment as an adversary to allow for synthesis of policies that are robust wrt. worst case performance or differences in simulations used for learning and real world application domains [54, 44, 53, 39]. These approaches optimize for worst-case robustness, but do not provide formal statistical guarantees on the resulting policies.

Another important line of research deals with the quantification of uncertainty and robustness to out-of-distribution data in reinforcement learning, enabling systems to identify their “known unknowns” [61, 48]. This is important also from a verification perspective, as any verification results achieved before system execution are valid if the data distribution stays the same at runtime.

## 6 Limitations

While SNES is able to incorporate information from the learning process into the synthesis of feasible policies given probabilistic constraints as requirements, there are a number of limitations to be aware of.

**Feedback loops and non-stationary data** As the policy is changing in the course of learning, the estimation of satisfaction probability and confidence therein is done on data that is generated by a non-stationary process. In the other direction, the current estimate is used by SNES to update the policy, thus creating a feedback loop. Therefore, the estimates made by SNES while learning are to be interpreted with care: The degree of non-stationarity may severely influence the validity of the estimates. This does however not affect a posteriori verification results, which are obtained for stationary CMDP and policy.

**Lack of convergence proof** While our current approach leveraging the Beta distribution for adaptively adjusting the Lagrangian yields interesting and effective results empirically, SNES lacks rigorous proofs of convergence and local optimality so far. We consider this a relevant direction for future work.

**Bounded verification** In its current formulation, SNES performs bounded verification for a given horizon (i.e. episode length). It is unclear how to interpret or model probabilistic system requirements and satisfaction for temporally unbound systems, as in the limit every possible event will occur almost surely. A promising direction could be the integration of rates as usually performed in Markov chain analysis, or to resort to average reward formulations of reinforcement learning [50]. Another approach could involve learning accepting sets of Büchi automata based on the MDP structure with non-determinism resolved by the current policy.

**No termination criterion** PSyCo combines optimization goals with constraints. While it is possible to decide whether constraints are satisfied, or at least to quantify confidence in the matter, it is usually not possible to decide whether the optimization goal has been reached or not. One approach to this would be to formulate requirements wrt. reward as constraints as well, such as requiring the system to reach a certain reward threshold [14]. In this case, policy synthesis could terminate when all given requirements are satisfied with a certain confidence.

## 7 Conclusion

We proposed to leverage epistemic uncertainty about constraint satisfaction of a reinforcement learner in safety critical domains. We introduced *Policy Synthesis under probabilistic Constraints* (PSyCo), a framework for specification of requirements for reinforcement learners in constrained settings, including confidence about results. PSyCo is organized along the classical phases of systematic software development: a system specification comprising a constrained Markov decision process as domain specification and a requirement specification in terms of probabilistic constraints, an abstract design defined by an algorithm for safe

policy synthesis with reinforcement learning and Bayesian model checking for system verification.

As an implementation of PSyCo we introduced *Safe Neural Evolutionary Strategies* (SNES), a method for learning safe policies under probabilistic constraints. SNES is leveraging online Bayesian model checking to obtain estimates of constraint satisfaction probability and a confidence in this estimate. SNES uses the confidence estimate to weight return and cost adaptively in a principled way in order to provide a sensible optimization target wrt. the constrained task. SNES provides a way to synthesize policies that are likely to satisfy a given specification.

We have empirically evaluated SNES in a sample domain designed to show the potentially interfering optimization goals of maximizing return while reaching and maintaining constraint satisfaction. We have shown that SNES is able to synthesize policies that are very likely to satisfy probabilistic constraints.

We see various directions for future research in safe system and policy synthesis. As a direct extension to our work, it would be interesting to extend other reinforcement learning algorithms with our approach of online adaptation of the Lagrangian with Bayesian model checking, such as value-based, actor-critic and policy gradient algorithms. We also think that notions for unbound probabilistic verification are of high interest for policy synthesis with general safety properties. Another direction could be the inclusion of curricula into the learning process, gradually increasing the strength of the constraints over the course of learning, thus potentially speeding up the learning process and allowing for convergence to more effective local optima. Finally, we think that safe learning in multi-agent systems dealing with feedback loops, strategic decision making and non-stationary learning dynamics poses interesting challenges for future research.

**Acknowledgements** We thank the anonymous reviewers for their constructive criticisms and helpful suggestions.

## A Finite PCTL for MDPs

We adapt probabilistic computation tree logic (PCTL) to finite sequences yielding a suitable logic to express safety constraints in our setup, allowing to specify constraints on satisfaction probabilities as well as bounding costs that may arise in system execution [55, 11].

In our approach, a PCTL constraint is of the form  $\mathbb{P}_J(\varphi)$  and specifies a bound  $J \subseteq [0, 1]$  on the probability that  $\varphi$  holds. The path formula  $\varphi$  consists of a single modal operator  $\bigcirc$ ,  $U$ ,  $U^{\leq m}$ ,  $\square$ , or  $\diamond$  and propositional state formulas as arguments. More formally, the path formula  $\varphi$  is formed according to the following syntax.

$$\varphi = \bigcirc\phi \mid \phi_1 U \phi_2 \mid \phi_1 U^{\leq m} \phi_2 \mid \square\phi \mid \diamond\phi \quad (40)$$

Here  $m \in \mathbb{N}$  and  $\phi, \phi_1, \phi_2$  are propositional state formulae built over atomic formulas using the constant true and the propositional connectives  $\neg, \wedge,$  and  $\vee$ .

We interpret PCTL path formulas as bounded by the length of an episode and define the semantics by a satisfaction relation  $\models_{\leq n}$  over finite episodes of a fixed length  $n$  where w.l.o.g. we assume  $n \geq \max(m, 2)$ .

The semantics  $\vec{e} \models_{\leq n} \varphi$  is defined as follows for finite sequences  $\vec{e} = e_1, \dots, e_n$  of episodes of length  $n$  with  $e_i = (s_{i-1}, a_{i-1}, s_i, r_i, c_i)$  for  $i = 1, \dots, n$  and  $|\vec{e}| = n$ .

$$\vec{e} \models_{\leq n} \bigcirc \phi \iff s_1 \models \phi \quad (41)$$

$$\vec{e} \models_{\leq n} \phi_1 U \phi_2 \iff \exists j, 0 \leq j \leq n : s_j \models \phi_2 \wedge \forall k, 0 \leq k < j : s_k \models \phi_1 \quad (42)$$

$$\vec{e} \models_{\leq n} \phi_1 U^{\leq m} \phi_2 \iff \exists j, 0 \leq j \leq m : s_j \models \phi_2 \wedge \forall k, 0 \leq k < j : s_k \models \phi_1 \quad (43)$$

$$\vec{e} \models_{\leq n} \Box \phi \iff \forall k, 0 \leq k \leq n : s_k \models \phi \quad (44)$$

The satisfaction relation  $s \models \phi$  for propositional formulas  $\phi$  in state  $s$  is defined as usual. The semantics of the probability operator is given by the probability measure  $Pr_s$  of all episodes of length  $n$  starting at state  $s$  and satisfying a path formula  $\varphi$ .

$$s_0 \models \mathbb{P}_J(\varphi) \text{ if } Pr_{s_0} \{ \vec{e} \in \text{Episodes}_n : \vec{e} \models_{\leq n} \varphi \} \in J \quad (45)$$

As usual we define  $\diamond \phi =_{\text{def}} \text{true } U \phi$ . Then the  $\Box$  modality can be expressed as follows (see e.g. [9]):  $\mathbb{P}_J(\Box \phi) =_{\text{def}} \mathbb{P}_{[0,1] \setminus J}(\diamond \neg \phi)$ .

This semantics coincides with the usual PCTL semantics over infinite paths. To show this we consider infinite sequences  $e^{\infty}$  of episodes and their infinite sequences  $s_0, s_1, s_2, \dots$  of states which we denote by  $s(e^{\infty})$ . Let  $e^{\infty}|_n$  denote the prefix (or starting sequence) of length  $n$  of  $e^{\infty}$ .

We define the  $n$ -bounded relativization  $\varphi^{\leq n}$  of a path formula  $\varphi$  (with  $n \geq \max\{2, m\}$ ) as follows:

$$(\bigcirc \phi)^{\leq n} =_{\text{def}} \bigcirc \phi \quad (46)$$

$$(\phi_1 U \phi_2)^{\leq n} =_{\text{def}} \phi_1 U^{\leq n} \phi_2 \quad (47)$$

$$(\phi_1 U^{\leq m} \phi_2)^{\leq n} =_{\text{def}} \phi_1 U^{\leq m} \phi_2 \quad (48)$$

$$(\Box \phi)^{\leq n} =_{\text{def}} \Box^{\leq n} \phi \quad (49)$$

$$(\diamond \phi)^{\leq n} =_{\text{def}} \text{true } U^{\leq n} \phi \quad (50)$$

where the auxiliary box modality  $\Box^{\leq n}$  is semantically defined by

$$s_0, s_1, s_2, \dots \models \Box^{\leq n} \phi \iff \forall k, 0 \leq k \leq n : s_k \models \phi \quad (51)$$

For any path formula, the semantics over finite episodes of length  $n$  coincides with the standard PCTL semantics of its  $n$ -bounded relativization; since we consider only bounded path formulas, the same holds for a leading probability operator:

**Fact 1** For any infinite sequence  $e^\infty$  of episodes with initial state  $s_0$ , any path formula  $\varphi$ , and any  $n$  with  $n \geq \max\{2, m + 1\}$ , the following holds:

$$s(e^\infty) \models \varphi^{\leq n} \iff e^\infty|_n \models_{\leq n} \varphi \quad (52)$$

$$s_0 \models \mathbb{P}_J(\varphi^{\leq n}) \iff s_0 \models_{\leq n} \mathbb{P}_J(\varphi) \quad (53)$$

## B Cost Function and Cumulative Cost

For any propositional state formula  $\phi$  we define a cost function  $C_\phi : S \times A \times S \rightarrow \mathbb{R}$  for a given CMDP for all  $a \in A, s, s' \in S$ .

$$C_\phi(s, a, s') = \begin{cases} 0 & \text{if } s' \models \phi \\ 1 & \text{otherwise} \end{cases} \quad (54)$$

$C_\phi$  assigns a cost for the value of  $\phi$  in the post state of any transition.

The cost of the initial state of an episode can be computed with the function  $c_\phi : S \rightarrow \mathbb{R}$  defined by

$$c_\phi(s) = \begin{cases} 0 & \text{if } s \models \phi \\ 1 & \text{otherwise} \end{cases} \quad (55)$$

Let  $\vec{e}$  be an episode of length  $n$ ,  $m \leq n$ , and  $\vec{s} = s_0, \dots, s_n = s_0 \circ \vec{s}'$  denote the path  $s(\vec{e})$  of that episode.

For any path  $\vec{s}$ , the cumulative cost function  $\mathcal{C}_{\square\phi} : Path_n \rightarrow \mathbb{R}$  of path formula  $\square\phi$  counts the number of violations of the state formula  $\phi$  in  $\vec{s}$ ; the cumulative cost is 0 if there are no violations, if  $\square\phi$  holds for  $\vec{e}$ . We give here a specification of  $\mathcal{C}_{\square\phi}$  based on the recursively defined cost of the corresponding bounded path formula  $\mathcal{C}_{\square^{\leq n}\phi}$ .

$$\mathcal{C}_{\square\phi, \vec{s}} = \mathcal{C}_{\square^{\leq n}\phi, \vec{s}} \quad (56)$$

$$\mathcal{C}_{\square^{\leq m}\phi, \vec{s}} = c_\phi(s_0) + \mathcal{C}'_{\square^{\leq m}\phi, \vec{s}} \quad (57)$$

$$\mathcal{C}'_{\square^{\leq m}\phi, \vec{s}} = \begin{cases} 0 & \text{if } m = 0 \\ C_\phi(s_0, a_0, s_1) + \mathcal{C}'_{\square^{\leq m-1}\phi, \vec{s}} & \text{if } m > 0 \end{cases} \quad (58)$$

Clearly definition 56 is equivalent to the sum in equation 10. The cumulative of  $\bigcirc\phi$  is the cost of the first action of the episode. For  $\phi_1 U \phi_2$  and  $\phi_1 U^{\leq m} \phi_2$  we count the number of violations of  $\phi_1$  and possibly the last violation of  $\phi_2$ .

$$\mathcal{C}_{\bigcirc\phi, \vec{s}} = C_\phi(s_0, a_0, s_1) \quad (59)$$

$$\mathcal{C}_{\phi_1 U \phi_2, \vec{s}} = \mathcal{C}_{\phi_1 U^{\leq n} \phi_2, \vec{s}} \quad (60)$$

$$\mathcal{C}_{\phi_1 U^{\leq m} \phi_2, \vec{s}} = c_{\phi_2}(s_0) (c_{\phi_1}(s_0) + \mathcal{C}'_{\phi_1 U^{\leq m} \phi_2, \vec{s}}) \quad (61)$$

$$\mathcal{C}'_{\phi_1 U^{\leq m} \phi_2, \vec{s}} = \begin{cases} 1 & \text{if } m = 0 \\ C_{\phi_2}(s_0, a_0, s_1) (C_{\phi_1}(s_0, a_0, s_1) + \mathcal{C}'_{\phi_1 U^{\leq m-1} \phi_2, \vec{s}}) & \text{if } m > 0 \end{cases} \quad (62)$$

The cumulative costs of a path constraint are 0 if the path constraint holds for the episode.

**Fact 2** *Let  $\vec{e}$  be any episode and  $\varphi$  any path formula. Then the following holds:*

$$\vec{e} \models \varphi \text{ iff } \mathcal{C}_{\varphi, s(\vec{e})} = 0 \quad (63)$$

## References

- [1] D. Abeywickrama, M. Mamei, and F. Zambonelli. The SOTA approach to engineering collective adaptive systems. *International Journal on Software Tools for Technology Transfer*, 1, 2020.
- [2] Dhaminda B. Abeywickrama and Franco Zambonelli. Model checking goal-oriented requirements for self-adaptive systems. In Miroslav Popovic, Bernhard Schätz, and Sebastian Voss, editors, *IEEE 19th International Conference and Workshops on Engineering of Computer-Based Systems, ECBS 2012, Novi Sad, Serbia, April 11-13, 2012*, pages 33–42. IEEE Computer Society, 2012.
- [3] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.
- [4] Eitan Altman. *Constrained Markov decision processes*, volume 7. CRC Press, 1999.
- [5] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*, 2016.
- [6] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.
- [7] Guy Avni, Roderick Bloem, Krishnendu Chatterjee, Thomas A Henzinger, Bettina Könighofer, and Stefan Pranger. Run-time optimization for learned controllers through quantitative games. In *International Conference on Computer Aided Verification*, pages 630–649. Springer, 2019.
- [8] Edoardo Bacci and D. Parker. Probabilistic guarantees for safe deep reinforcement learning. In *Proceedings of the 18th International Conference on Formal Modeling and Analysis of Timed Systems*, pages 231–248, 2020.
- [9] Christel Baier, Luca de Alfaro, Vojtech Forejt, and Marta Kwiatkowska. Model checking probabilistic systems. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 963–999. Springer, 2018.

- [10] Christel Baier, Holger Hermanns, and Joost-Pieter Katoen. The 10,000 facets of MDP model checking. In *Computing and Software Science*, pages 420–451. Springer, 2019.
- [11] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT press, 2008.
- [12] Jorge A. Baier and Sheila A. McIlraith. Planning with preferences. *AI Mag.*, 29(4):25–36, 2008.
- [13] Brian Beavis and Ian Dobbs. *Optimisation and Stability theory for Economic Analysis*. Cambridge University Press, 1990.
- [14] Lenz Belzner and Thomas Gabor. QoS-aware multi-armed bandits. In *2016 IEEE 1st International Workshops on Foundations and Applications of Self\* Systems (FAS\* W)*, pages 118–119. IEEE, 2016.
- [15] Lenz Belzner and Thomas Gabor. Bayesian verification under model uncertainty. In *2017 IEEE/ACM 3rd International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS)*, pages 10–13. IEEE, 2017.
- [16] Lenz Belzner, Rolf Hennicker, and Martin Wirsing. Onplan: A framework for simulation-based online planning. In Christiano Braga and Peter Csaba Ölveczky, editors, *Formal Aspects of Component Software - 12th International Conference, FACS 2015, Niterói, Brazil, October 14-16, 2015, Revised Selected Papers*, volume 9539 of *Lecture Notes in Computer Science*, pages 1–30. Springer, 2015.
- [17] Suda Bharadwaj, Roderik Bloem, Rayna Dimitrova, Bettina Könighofer, and Ufuk Topcu. Synthesis of minimum-cost shields for multi-agent systems. In *2019 American Control Conference (ACC)*, pages 1048–1055. IEEE, 2019.
- [18] Tomás Bures, Rocco De Nicola, Ilias Gerostathopoulos, Nicklas Hoch, Michal Kit, Nora Koch, Giacomina Valentina Monreale, Ugo Montanari, Rosario Pugliese, Nikola B. Serbedzija, Martin Wirsing, and Franco Zambonelli. A life cycle for the development of autonomic systems: The e-mobility showcase. In *7th IEEE International Conference on Self-Adaptation and Self-Organizing Systems Workshops, SASO, 2013, Philadelphia, PA, USA, September 9-13, 2013*, pages 71–76. IEEE Computer Society, 2013.
- [19] Rui P. Cardoso, Rosaldo J. F. Rossetti, Emma Hart, David Burth Kurka, and Jeremy Pitt. Engineering sustainable and adaptive systems in dynamic and unpredictable environments. In Margaria and Steffen [52], pages 221–240.

- [20] Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. Risk-constrained reinforcement learning with percentile risk criteria. *The Journal of Machine Learning Research*, 18(1):6070–6120, 2017.
- [21] Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. A Lyapunov-based approach to safe reinforcement learning. In *Advances in neural information processing systems*, pages 8092–8101, 2018.
- [22] Edmund M Clarke and Paolo Zuliani. Statistical model checking for cyber-physical systems. In *International Symposium on Automated Technology for Verification and Analysis*, pages 1–12. Springer, 2011.
- [23] Carlos A. Coello Coello. Multi-objective optimization. In Rafael Martí, Panos M. Pardalos, and Mauricio G. C. Resende, editors, *Handbook of Heuristics*, pages 177–204. Springer, 2018.
- [24] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Sci. Comput. Program.*, 20(1-2):3–50, 1993.
- [25] Persi Diaconis, Donald Ylvisaker, et al. Conjugate priors for exponential families. *The Annals of Statistics*, 7(2):269–281, 1979.
- [26] Iulia Dragomir, Simon Iosti, Marius Bozga, and Saddek Bensalem. Designing systems with detection and reconfiguration capabilities: A formal approach. In Margaria and Steffen [52], pages 155–171.
- [27] Madalina M. Drugan. Multi-objective optimization perspectives on reinforcement learning algorithms using reward vectors. In *ESANN*, 2015.
- [28] Jiameng Fan and Wenchao Li. Safety-guided deep reinforcement learning via online Gaussian process estimation. *arXiv preprint arXiv:1903.02526*, 2019.
- [29] Nathan Fulton and André Platzer. Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [30] Nathan Fulton and André Platzer. Verifiably safe off-model reinforcement learning. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 413–430. Springer, 2019.
- [31] Javier Garcia and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.
- [32] Yangyang Ge, Fei Zhu, Xinghong Ling, and Quan Liu. Safe Q-learning method based on constrained Markov decision processes. *IEEE Access*, 7:165007–165017, 2019.

- [33] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.
- [34] Sofie Haesaert, Sadegh Soudjani, and Alessandro Abate. Temporal logic control of general Markov decision processes by approximate policy refinement. *IFAC-PapersOnLine*, 51(16):73–78, 2018.
- [35] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Cautious reinforcement learning with logical constraints. In *Proceedings of the 19th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 483–491, 2020.
- [36] Mohammadhosein Hasanbeig, Daniel Kroening, and Alessandro Abate. Deep reinforcement learning with temporal logics. In *Proceedings of the 18th International Conference on Formal Modeling and Analysis of Timed Systems*, pages 1–22, 2020.
- [37] Matthias M. Hözl and Thomas Gabor. Continuous collaboration for changing environments. *Trans. Found. Mastering Chang.*, 1:201–224, 2016.
- [38] Matthias M. Hözl, Nora Koch, Mariachiara Puviani, Martin Wirsing, and Franco Zambonelli. The ensemble development life cycle and best practices for collective autonomic systems. In Wirsing et al. [63], pages 325–354.
- [39] Manfred Jaeger, Giorgio Bacci, Giovanni Bacci, Kim Guldstrand Larsen, and Peter Gjøøl Jensen. Approximating euclidean by imprecise markov decision processes. In *International Symposium on Leveraging Applications of Formal Methods*, pages 275–289. Springer, 2020.
- [40] Nils Jansen, Bettina Könighofer, Sebastian Junges, Alex Serban, and Roderick Bloem. Safe Reinforcement Learning Using Probabilistic Shields (Invited Paper). In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory (CONCUR 2020)*, volume 171 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:16, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.
- [41] Sumit K Jha, Edmund M Clarke, Christopher J Langmead, Axel Legay, André Platzer, and Paolo Zuliani. A Bayesian approach to model checking biological systems. In *International Conference on Computational Methods in Systems Biology*, pages 218–234. Springer, 2009.
- [42] Sebastian Junges, Nils Jansen, Christian Dehnert, Ufuk Topcu, and Joost-Pieter Katoen. Safety-constrained reinforcement learning for MDPs. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 130–146. Springer, 2016.
- [43] Sebastian Junges, Nils Jansen, and Sanjit A Seshia. Enforcing almost-sure reachability in POMDPs. *arXiv preprint arXiv:2007.00085*, 2020.

- [44] Richard Klima, Daan Bloembergen, Michael Kaisers, and Karl Tuyls. Robust temporal difference learning for critical domains. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 350–358. International Foundation for Autonomous Agents and Multiagent Systems, 2019.
- [45] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In *International Conference on Computer Aided Verification*, pages 585–591. Springer, 2011.
- [46] Kim Guldstrand Larsen and Axel Legay. Statistical model checking the 2018 edition! In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Verification - 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part II*, volume 11245 of *Lecture Notes in Computer Science*, pages 261–270. Springer, 2018.
- [47] Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: An overview. In *International Conference on Runtime Verification*, pages 122–135. Springer, 2010.
- [48] Björn Lötjens, Michael Everett, and Jonathan P How. Safe reinforcement learning with model uncertainty estimates. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8662–8668. IEEE, 2019.
- [49] Jeff Magee and Jeff Kramer. *Concurrency - State Models and Java Programs (2. ed.)*. Wiley, 2006.
- [50] Sridhar Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine learning*, 22(1-3):159–195, 1996.
- [51] Housseem Ben Mahfoudh, Giovanna Di Marzo Serugendo, Anthony Boulmier, and Nabil Abdennadher. Coordination model with reinforcement learning for ensuring reliable on-demand services in collective adaptive systems. In Margaria and Steffen [52], pages 257–273.
- [52] Tiziana Margaria and Bernhard Steffen, editors. *Leveraging Applications of Formal Methods, Verification and Validation. Distributed Systems - 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part III*, volume 11246 of *Lecture Notes in Computer Science*. Springer, 2018.
- [53] Thomy Phan, Thomas Gabor, Andreas Sedlmeier, Fabian Ritz, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Jan Wieghardt, Marc Zeller, and Claudia Linnhoff-Popien. Learning and testing resilience in cooperative multi-agent systems. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2020.

- [54] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2817–2826. JMLR. org, 2017.
- [55] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57. IEEE, 1977.
- [56] Alex Ray, Joshua Achiam, and Dario Amodei. Benchmarking safe exploration in deep reinforcement learning. Technical report, Open AI, 2019.
- [57] Diederik M. Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *J. Artif. Intell. Res.*, 48:67–113, 2013.
- [58] Stuart Russell, Daniel Dewey, and Max Tegmark. Research priorities for robust and beneficial artificial intelligence. *AI Magazine*, 36(4):105–114, 2015.
- [59] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.
- [60] Dirk Schäfer and Eyke Hüllermeier. Preference-based reinforcement learning using dyad ranking. In Larisa N. Soldatova, Joaquin Vanschoren, George A. Papadopoulos, and Michelangelo Ceci, editors, *Discovery Science - 21st International Conference, DS 2018, Limassol, Cyprus, October 29-31, 2018, Proceedings*, volume 11198 of *Lecture Notes in Computer Science*, pages 161–175. Springer, 2018.
- [61] Andreas Sedlmeier, Thomas Gabor, Thomy Phan, Lenz Belzner, and Claudia Linnhoff-Popien. Uncertainty-based out-of-distribution classification in deep reinforcement learning. *arXiv preprint arXiv:2001.00496*, 2019.
- [62] Bernd Waschneck, André Reichstaller, Lenz Belzner, Thomas Altmüller, Thomas Bauernhansl, Alexander Knapp, and Andreas Kyek. Optimization of global production scheduling with deep reinforcement learning. *Procedia CIRP*, 72(1):1264–1269, 2018.
- [63] Martin Wirsing, Matthias M. Hözl, Nora Koch, and Philip Mayer, editors. *Software Engineering for Collective Autonomic Systems - The ASCENS Approach*, volume 8998 of *Lecture Notes in Computer Science*. Springer, 2015.
- [64] Martin Wirsing, Matthias M. Hözl, Mirco Tribastone, and Franco Zambonelli. ASCENS: engineering autonomic service-component ensembles. In Bernhard Beckert, Ferruccio Damiani, Frank S. de Boer, and Marcello M. Bonsangue, editors, *FMCO 2011, Revised Selected Papers*, volume 7542 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2011.

- [65] Paolo Zuliani, André Platzer, and Edmund M Clarke. Bayesian statistical model checking with application to Simulink/Stateflow verification. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, pages 243–252. ACM, 2010.