Safety-Aware Apprenticeship Learning

Weichao Zhou and Wenchao Li

Department of Electrical and Computer Engineering
Boston University
{zwc662, wenchao}@bu.edu

Abstract. Apprenticeship learning (AL) is a kind of Learning from Demonstration techniques where the reward function of a Markov Decision Process (MDP) is unknown to the learning agent and the agent has to derive a good policy by observing an expert's demonstrations. In this paper, we study the problem of how to make AL algorithms inherently safe while still meeting its learning objective. We consider a setting where the unknown reward function is assumed to be a linear combination of a set of state features, and the safety property is specified in Probabilistic Computation Tree Logic (PCTL). By embedding probabilistic model checking inside AL, we propose a novel *counterexample-guided* approach that can ensure safety while retaining performance of the learnt policy. We demonstrate the effectiveness of our approach on several challenging AL scenarios where safety is essential.

1 Introduction

The rapid progress of artficial intelligence (AI) comes with a growing concern over its safety when deployed in real-life systems and situations. As highlighted in [3], if the objective function of an AI agent is wrongly specified, then maximizing that objective function may lead to harmful results. In addition, the objective function or the training data may focus only on accomplishing a specific task and ignore other aspects, such as safety constraints, of the environment. In this paper, we propose a novel framework that combines explicit safety specification with learning from data. We consider safety specification expressed in Probabilistic Computation Tree Logic (PCTL) and show how probabilistic model checking can be used to ensure safety and retain performance of a learning algorithm known as *apprenticeship learning* (AL).

We consider the formulation of apprenticeship learning by Abbeel and Ng [1]. The concept of AL is closely related to *reinforcement learning* (RL) where an agent learns what actions to take in an environment (known as a policy) by maximizing some notion of long-term reward. In AL, however, the agent is not given the reward function, but instead has to first estimate it from a set of expert demonstrations via a technique called *inverse reinforcement learning* [18] . The formulation assumes that the reward function is expressible as a linear combination of *known state features*. An expert demonstrates the task by maximizing this reward function and the agent tries to derive a policy that can

match the feature expectations of the expert's demonstrations. Apprenticeship learning can also be viewed as an instance of the class of techniques known as Learning from Demonstration (LfD). One issue with LfD is that *the expert often can only demonstrate how the task works but not how the task may fail.* This is because failure may cause irrecoverable damages to the system such as crashing a vehicle. In general, the lack of "negative examples" can cause a heavy bias in how the learning agent constructs the reward estimate. In fact, *even if all the demonstrations are safe, the agent may still end up learning an unsafe policy.*

The key idea of this paper is to incorporate formal verification in apprenticeship learning. We are inspired by the line of work on formal inductive synthesis [10] and counterexample-guided inductive synthesis [22]. Our approach is also similar in spirit to the recent work on safety-constrained reinforcement learning [11]. However, our approach uses the results of model checking in a novel way. We consider safety specification expressed in probabilistic computation tree logic (PCTL). We employ a verification-in-the-loop approach by embedding PCTL model checking as a safety checking mechanism inside the learning phase of AL. In particular, when a learnt policy does not satisfy the PCTL formula, we leverage counterexamples generated by the model checker to steer the policy search in AL. In essence, counterexample generation can be viewed as supplementing negative examples for the learner. Thus, the learner will try to find a policy that not only imitates the expert's demonstrations but also stays away from the failure scenarios as captured by the counterexamples.

In summary, we make the following contributions in this paper.

- We propose a novel framework for incorporating formal safety guarantees in Learning from Demonstration.
- We develop a novel algorithm called CounterExample Guided Apprenticeship Learning (CEGAL) that combines probabilistic model checking with the optimization-based approach of apprenticeship learning.
- We demonstrate that our approach can guarantee safety for a set of case studies and attain performance comparable to that of using apprenticeship learning alone.

The rest of the paper is organized as follows. Section 2 reviews background information on apprenticeship learning and PCTL model checking. Section 3 defines the safety-aware apprenticeship learning problem and gives an overview of our approach. Section 4 illustrates the counterexample-guided learning framework. Section 5 describes the proposed algorithm in detail. Section 6 presents a set of experimental results demonstrating the effectiveness of our approach. Section 7 discusses related work. Section 8 concludes and offers future directions.

2 Preliminaries

2.1 Markov Decision Process and Discrete-Time Markov Chain

Markov Decision Process (MDP) is a tuple $M = (S, A, T, \gamma, s_0, R)$, where S is a finite set of states; A is a set of actions; $T : S \times A \times S \to [0, 1]$ is

a transition function describing the probability of transitioning from one state $s \in S$ to another state by taking action $a \in A$ in state s; $R : S \to \mathbb{R}$ is a reward function which maps each state $s \in S$ to a real number indicating the reward of being in state s; $s_0 \in S$ is the initial state; $\gamma \in [0, 1)$ is a discount factor which describes how future rewards attenuate when a sequence of transitions is made. A deterministic and stationary (or memoryless) policy $\pi : S \to A$ for an MDP M is a mapping from states to actions, i.e. the policy deterministically selects what action to take solely based on the current state. In this paper, we restrict ourselves to deterministic and stationary policy. A policy π for an MDP M induces a Discrete-Time Markov Chain (DTMC) $M_\pi = (S, T_\pi, s_0)$, where $T_\pi : S \times S \to [0, 1]$ is the probability of transitioning from a state s to another state in one step. A trajectory $\tau = s_0 \xrightarrow{T_\pi(s_0,s_1)>0} s_1 \xrightarrow{T_\pi(s_1,s_2)>0} s_2, ...,$ is a sequence of states where $s_i \in S$. The accumulated reward of τ is $\sum_{i=0}^{\infty} \gamma^i R(s_i)$. The value function $V_\pi : S \to \mathbb{R}$ measures the expectation of accumulated reward $E[\sum_{i=0}^{\infty} \gamma^i R(s_i)]$ starting from a state s and following policy π. An *optimal policy* π for MDP M is a policy that maximizes the value function [4].

2.2 Apprenticeship Learning via Inverse Reinforcement Learning

Inverse reinforcement learning (IRL) aims at recovering the reward function R of $M \backslash R = (S, A, T, \gamma, s_0)$ from a set of m trajectories $\Gamma_E = \{\tau_0, \tau_1, ..., \tau_{m-1}\}$ demonstrated by an expert. *Apprenticeship learning (AL)* [1] assumes that the reward function is a linear combination of state features, i.e. $R(s) = \omega^T f(s)$ where $f : S \to [0,\ 1]^k$ is a vector of known features over states S and $\omega \in \mathbb{R}^k$ is an unknown weight vector that satisfies $||\omega||_2 \leq 1$. The expected features of a policy π are the expected values of the cumulative discounted state features $f(s)$ by following π on M, i.e. $\mu_\pi = E[\sum_{t=0}^{\infty} \gamma^t f(s_t)|\pi]$. Let μ_E denote the expected features of the unknown expert's policy π_E. μ_E can be approximated by the expected features of expert's m demonstrated trajectories $\hat{\mu}_E = \frac{1}{m} \sum_{\tau \in \Gamma_E} \sum_{t=0}^{\infty} \gamma^t f(s_t)$ if m is large enough. With a slight abuse of notations, we use μ_Γ to also denote the expected features of a set of paths Γ. Given an error bound ϵ, a policy π^* is defined to be ϵ-*close* to π_E if its expected features μ_{π^*} satisfies $||\mu_E - \mu_{\pi^*}||_2 \leq \epsilon$. The expected features of a policy can be calculated by using Monte Carlo method, value iteration or linear programming [1,4].

The algorithm proposed by Abbeel and Ng [1] starts with a random policy π_0 and its expected features μ_{π_0}. Assuming that in iteration i, a set of i candidate policies $\Pi = \{\pi_0, \pi_1, ..., \pi_{i-1}\}$ and their corresponding expected features $\{\mu_\pi | \pi \in \Pi\}$ have been found, the algorithm solves the following optimization problem.

$$\delta = \max_{\omega} \min_{\pi \in \Pi} \ \omega^T (\hat{\mu}_E - \mu_\pi) \qquad s.t.\ ||\omega||_2 \leq 1 \tag{1}$$

The optimal ω is used to find the corresponding optimal policy π_i and the expected features μ_{π_i}. If $\delta \leq \epsilon$, then the algorithm terminates and π_i is produced

as the output. Otherwise, μ_{π_i} is added to the set of features for the candidate policy set Π and the algorithm continues to the next iteration.

2.3 PCTL Model Checking

Probabilistic model checking can be used to verify properties of a stochastic system such as "is the probability that the agent reaches the unsafe area within 10 steps smaller than 5%?". *Probabilistic Computation Tree Logic* (PCTL) [7] allows for probabilistic quantification of properties. The syntax of PCTL includes state formulas and path formulas [13]. A state formula ϕ asserts property of a single state $s \in S$ whereas a path formula ψ asserts property of a trajectory.

$$\phi ::= true \mid l_i \mid \neg\phi_i \mid \phi_i \wedge \phi_j \mid P_{\bowtie p^*}[\psi] \tag{2}$$

$$\psi ::= \mathbf{X}\phi \mid \phi_1 \mathbf{U}^{\leq k} \phi_2 \mid \phi_1 \mathbf{U} \phi_2 \tag{3}$$

where l_i is atomic proposition and ϕ_i, ϕ_j are state formulas; $\bowtie \in \{\leq, \geq, <, >\}$; $P_{\bowtie p^*}[\psi]$ means that the probability of generating a trajectory that satisfies formula ψ is $\bowtie p^*$. $\mathbf{X}\phi$ asserts that the next state after initial state in the trajectory satisfies ϕ; $\phi_1 \mathbf{U}^{\leq k} \phi_2$ asserts that ϕ_2 is satisfied in at most k transitions and all preceding states satisfy ϕ_1; $\phi_1 \mathbf{U} \phi_2$ asserts that ϕ_2 will be eventually satisfied and all preceding states satisfy ϕ_1. The semantics of PCTL is defined by a satisfaction relation $\models$ as follows.

$$s \models true \quad \text{iff state } s \in S \tag{4}$$

$$s \models \phi \quad \text{iff state s satisfies the state formula } \phi \tag{5}$$

$$\tau \models \psi \quad \text{iff trajectory } \tau \text{ satisfies the path formula } \psi. \tag{6}$$

Additionally, $\models_{min}$ denotes the minimal satisfaction relation [6] between τ and ψ. Defining $pref(\tau)$ as the set of all prefixes of trajectory τ including τ itself, then $\tau \models_{min} \psi$ iff $(\tau \models \psi) \wedge (\forall \tau' \in pref(\tau) \backslash \tau, \tau' \not\models \psi)$. For instance, if $\psi = \phi_1 \mathbf{U}^{\leq k} \phi_2$, then for any finite trajectory $\tau \models_{min} \phi_1 \mathbf{U}^{\leq k} \phi_2$, only the final state in τ satisfies ϕ_2. Let $P(\tau)$ be the probability of transitioning along a trajectory τ and let Γ_ψ be the set of all finite trajectories that satisfy $\tau \models_{min} \psi$, the value of PCTL property ψ is defined as $P_{=?|s_0}[\psi] = \sum_{\tau \in \Gamma_\psi} P(\tau)$. For a DTMC M_π and a state formula $\phi = P_{\leq p^*}[\psi]$, $M_\pi \models \phi$ iff $P_{=?|s_0}[\psi] \leq p^*$.

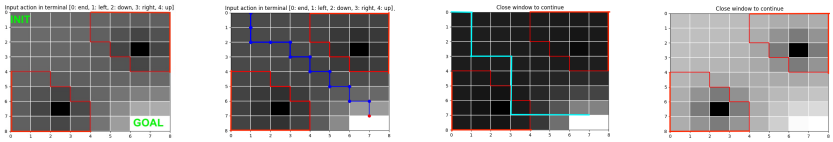
A *counterexample* of ϕ is a set $CEX \subseteq \Gamma_\psi$ that satisfies $\sum_{\tau \in CEX} P(\tau) > p^*$. Let $\mathbb{P}(\Gamma) = \sum_{\tau \in \Gamma} P(\tau)$ be the sum of probabilities of all trajectories in a set Γ. Let $CEX_\phi \subseteq 2^{\Gamma_\psi}$ be the set of all counterexamples for a formula ϕ such that $(\forall CEX \in CEX_\phi, \mathbb{P}(CEX) > p^*)$ and $(\forall \Gamma \in 2^{\Gamma_\psi} \backslash CEX_\phi, \mathbb{P}(\Gamma) \leq p^*)$. A *minimal counterexample* is a set $CEX \in CEX_\phi$ such that $\forall CEX' \in CEX_\phi, |CEX| \leq |CEX'|$. By converting DTMC M_π into a weighted directed

graph, counterexample can be found by solving a k-shortest paths (KSP) problem or a hop-constrained KSP (HKSP) problem [6]. Alternatively, counterexamples can be found by using Satisfiability Modulo Theory solving or mixed integer linear programming to determine the minimal critical subsystems that capture the counterexamples in M_π [23].

A policy can also be synthesized by solving the objective $\min_{\pi} P_{=?}[\psi]$ for an MDP M. This problem can be solved by linear programming or policy iteration (and value iteration for step-bounded reachability) [14].

3 Problem Formulation and Overview

Suppose there are some unsafe states in an $MDP\backslash R$ $M = (S, A, T, \gamma, s_0)$. A safety issue in apprenticeship learning means that an agent following the learnt policy would have a higher probability of entering those unsafe states than it should. There are multiple reasons that can give rise to this issue. First, it is possible that the expert policy π_E itself has a high probability of reaching the unsafe states. Second, human experts often tend to perform only successful demonstrations that do not highlight the unwanted situations [21]. This *lack of negative examples* in the training set can cause the learning agent to be unaware of the existence of those unsafe states.



(a) (b) (c) (d)

Fig. 1: The 8 x 8 grid-world. (a) Lighter grid cells have higher rewards than the darker ones. The two black grid cells have the lowest rewards, while the two white ones have the highest rewards. The grid cells enclosed by red lines are considered *unsafe*. (b) The blue line is an example trajectory demonstrated by the expert. (c) Only the goal states are assigned high rewards and there is little difference between the unsafe states and their nearby states. As a result, the learnt policy has a high probability of passing through the unsafe states as indicated by the cyan line. (d) $p^* = 20\%$. The learnt policy is optimal to a reward function that correctly assigns low rewards to the unsafe states.

We use a 8 x 8 grid-world navigation example as shown in Fig. 1 to illustrate this problem. An agent starts from the upper-left corner and moves from cell to cell until it reaches the lower-right corner. The 'unsafe' cells are enclosed by the red dashed lines. These represent regions that the agent should avoid. In each step, the agent can choose to stay in current cell or move to an adjacent cell but with 20% chance of moving randomly instead of following its decision. The goal area, the unsafe area and the reward mapping for all states are shown in Fig. 1(a). For each state $s \in S$, its feature vector consists of 4 radial basis feature

functions with respect to the squared Euclidean distances between s and the 4 states with the highest or lowest rewards as shown in Fig. 1(a). In addition, a specification Φ formalized in PCTL is used to capture the safety requirement. In (7), p^* is the required upper bound of the probability of reaching an unsafe state within $t = 64$ steps.

$$\Phi ::= P_{\leq p^*}[\texttt{true}\ \mathbf{U}^{\leq t}\ \texttt{unsafe}] \tag{7}$$

Let π_E be the optimal policy under the reward map shown in Fig. 1(a). The probability of entering an unsafe region within 64 steps by following π_E is 24.6%. Now consider the scenario where the expert performs a number of demonstrations by following π_E. *All demonstrated trajectories in this case successfully reach the goal areas without ever passing through any of the unsafe regions.* Fig. 1(b) shows a representative trajectory (in blue) among $10,000$ such demonstrated trajectories. The resulting reward map by running the AL algorithm on these $10,000$ demonstrations is shown in Fig. 1(c). Observe that only the goal area has been learnt whereas the agent is oblivious to the unsafe regions (treating them in the same way as other dark cells). In fact, the probability of reaching an unsafe state within 64 steps with this policy turns out to be 82.6% (thus violating the safety requirement by a large margin). To make matters worse, the value of p^* may be decided or revised after a policy has been learnt. In those cases, even the original expert policy π_E may be unsafe, e.g., when $p^* = 20\%$. Thus, we need to adapt the original AL algorithm so that it will take into account of such safety requirement. Fig. 1(d) shows the resulting reward map learned using our proposed algorithm (to be described in detail later) for $p^* = 20\%$. It clearly matches well with the color differentiation in the original reward map and captures both the goal states and the unsafe regions. This policy has an unsafe probability of 19.0%. We are now ready to state our problem.

Definition 1. ***The safety-aware apprenticeship learning (SafeAL) problem*** *is, given an $MDP\backslash R$, a set of m trajectories $\{\tau_0, \tau_1, ..., \tau_{m-1}\}$ demonstrated by an expert, and a specification Φ, to learn a policy π that satisfies Φ and is ϵ-close to the expert policy π_E.*

Remark 1. We note that a solution may not always exist for the SafeAL problem. While the decision problem of checking whether a solution exists is of theoretical interest, in this paper, we focus on tackling the problem of finding a policy π that satisfies a PCTL formula Φ (if Φ is satisfiable) and whose performance is as close to that of the expert's as possible, i.e. we relax the condition on μ_π being ϵ-close to μ_E.

4 A Framework for Safety-Aware Learning

In this section, we describe a general framework for safety-aware learning. This novel framework utilizes information from both the expert demonstrations and a verifier. The proposed framework is illustrated in Fig. 2. Similar to the *counterexample-guided inductive synthesis* (CEGIS) paradigm [22], our framework consists of a

verifier and a *learner*. The verifier checks if a candidate policy satisfies the safety specification Φ. In case Φ is not satisfied, the verifier generates a counterexample for Φ. The main difference from CEGIS is that our framework considers not only functional correctness, e.g., safety, but also performance (as captured by the learning objective). Starting from an initial policy π_0, each time the learner learns a new policy, the verifier checks if the specification is satisfied. If true, then this policy is added to the candidate set, otherwise the verifier will generate a (minimal) counterexample and add it to the counterexample set. During the learning phase, the learner uses both the counterexample set and candidate set to find a policy that is close to the (unknown) expert policy and far away from the counterexamples. The goal is to find a policy that is ϵ-close to the expert policy and satisfies the specification. For the grid-world example introduced in Section 3, when $p^* = 5\%$ (thus presenting a stricter safety requirement compared to the expert policy π_E), our approach produces a policy with only 4.2% of reaching an unsafe state within 64 steps (with the correspondingly inferred reward mapping shown in Fig. 1(d)).

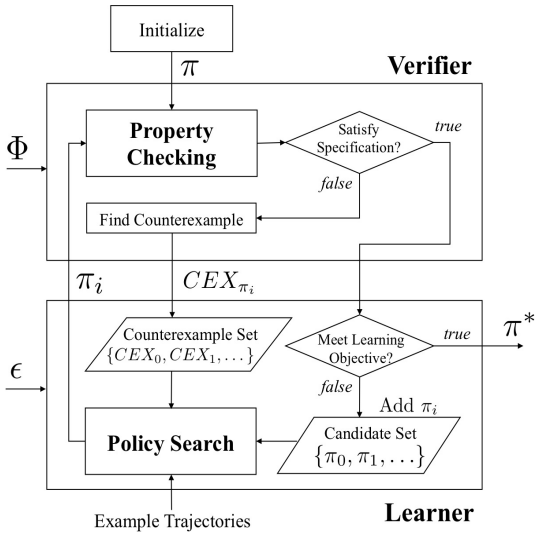
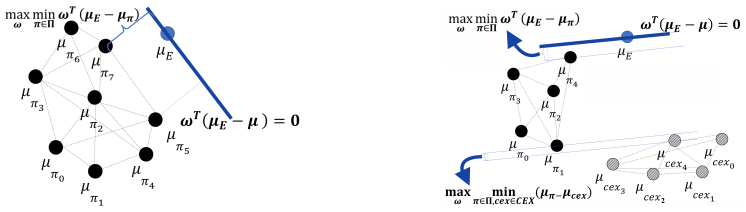


Fig. 2: Our safety-aware learning framework. Given an initial policy π_0, a specification Φ and a learning objective (as captured by ϵ), the framework iterates between a *verifier* and a *learner* to search for a policy π^* that satisfies both Φ and ϵ. One invariant that this framework maintains is that all the π_i's in the candidate policy set satisfy Φ.

Learning from a (minimal) counterexample cex_π of a policy π is similar to learning from expert demonstrations. The basic principle of the AL algorithm proposed in [1] is to find a weight vector ω under which the expected reward of π_E maximally outperforms any mixture of the policies in the candidate policy set $\Pi = \{\pi_0, \pi_1, \pi_2, \ldots\}$. Thus, ω can be viewed as the normal vector of the hyperplane $\omega^T(\mu - \mu_E) = 0$ that has the maximal distance to the convex hull of the set $\{\mu_\pi \mid \pi \in \Pi\}$ as illustrated in the 2D feature space in Fig. 3(a). It can be shown



(a) (b)

Fig. 3: (a) Learn from expert. (b) Learn from both expert demonstrations and counterexamples.

that $\omega^T \mu_\pi \geq \omega^T \mu_{\pi'}$ for all previously found π's. Intuitively, this helps to move the candidate μ_π closer to μ_E. Similarly, we can apply the same max-margin separation principle to maximize the distance between the candidate policies and the counterexamples (in the μ space). Let $CEX = \{cex_0, cex_1, cex_2, ...\}$ denote the set of counterexamples of the policies that do not satisfy the specification Φ. Maximizing the distance between the convex hulls of the sets $\{\mu_{cex} | cex \in CEX\}$ and $\{\mu_\pi \mid \pi \in \Pi\}$ is equivalent to maximizing the distance between the parallel supporting hyperplanes of the two convex hulls as shown in Fig. 3(b). The corresponding optimization function is given in Eq. (8).

$$\delta = \max_{\omega} \min_{\pi \in \Pi, cex \in CEX} \omega^T (\mu_\pi - \mu_{cex}) \qquad s.t. \; ||\omega||_2 \leq 1 \tag{8}$$

To attain good performance similar to that of the expert, we still want to learn from μ_E. Thus, the overall problem can be formulated as a multi-objective optimization problem that combines (1) and (8) into (9).

$$\max_{\omega} \min_{\pi \in \Pi, \tilde{\pi} \in \Pi, cex \in CEX} (\omega^T (\mu_E - \mu_\pi), \; \omega^T (\mu_{\tilde{\pi}} - \mu_{cex})) \qquad s.t. \; ||\omega||_2 \leq 1 \tag{9}$$

5 Counterexample-Guided Apprenticeship Learning

In this section, we introduce the CounterExample Guided Apprenticeship Learning (CEGAL) algorithm to solve the SafeAL problem. It can be viewed as a special case of the safety-aware learning framework described in the previous section. In addition to combining policy verification, counterexample generation and AL, our approach uses an adaptive weighting scheme to weight the separation from μ_E with the separation from μ_{cex}.

$$\begin{aligned} &\max_{\omega} \min_{\pi \in \Pi_S, \tilde{\pi} \in \Pi_S, cex \in CEX} \omega^T (k(\mu_E - \mu_\pi) + (1-k)(\mu_{\tilde{\pi}} - \mu_{cex})) \qquad (10) \\ &s.t. \; ||\omega||_2 \leq 1, \; k \in [0,1] \\ &\quad \omega^T (\mu_E - \mu_\pi) \leq \omega^T (\mu_E - \mu_{\pi'}), \; \forall \pi' \in \Pi_S \\ &\quad \omega^T (\mu_{\tilde{\pi}} - \mu_{cex}) \leq \omega^T (\mu_{\tilde{\pi}'} - \mu_{cex'}), \; \forall \tilde{\pi}' \in \Pi_S, \forall cex' \in CEX \end{aligned}$$

In essence, we take a weighted-sum approach for solving the multi-objective optimization problem (9). Assuming that $\Pi_S = \{\pi_1, \pi_2, \pi_3, \ldots\}$ is a set of candidate policies that all satisfy Φ, $CEX = \{cex_1, cex_2, cex_3, \ldots\}$ is a set of counterexamples. We introduce a parameter k and change (9) into a weighted sum optimization problem (10). Note that π and $\tilde{\pi}$ can be different. The optimal ω solved from (10) can be used to generate a new policy π_ω by using algorithms such as policy iteration. We use a probabilistic model checker, such as PRISM [13], to check if π_ω satisfies Φ. If it does, then it will be added to Π_S. Otherwise, a counterexample generator, such as COMICS [9], is used to generate a (minimal) counterexample cex_{π_ω}, which will be added to CEX.

Algorithm 1 Counterexample-Guided Apprenticeship Learning (CEGAL)

Input:
  $M \leftarrow$ A partially known $MDP \backslash R$; $f \leftarrow$ A vector of feature functions
  $\mu_E \leftarrow$ The expected features of expert trajectories $\{\tau_0, \tau_1, \ldots, \tau_m\}$
  $\Phi \leftarrow$ Specification; $\epsilon \leftarrow$ Error bound for the expected features;
  $\sigma, \alpha \in (0, 1) \leftarrow$ Error bound σ and step length α for the parameter k;
Initialization:
  $inf \leftarrow 0, sup \leftarrow 1, k \leftarrow sup$ ▷ Initialize multi-optimization parameter k
  $\Pi_S \leftarrow \{\pi_0\}$ ▷ Initialize candidate set with an **initial safe policy**
  $CEX \leftarrow \emptyset$ ▷ Initialize counterexample set as empty
  $\pi_1 \leftarrow$ Policy learnt from μ_E via apprenticeship learning
Iteration i $(i \geq 1)$:
  Verifier:
    status $\leftarrow Model\_Checker(M, \pi_i, \Phi)$
    If status = SAT, **then go to Learner**
    If status = UNSAT
      $cex_{\pi_i} \leftarrow Counterexample\_Generator(M, \pi_i, \Phi)$
      Add cex_{π_i} to CEX and solve $\mu_{cex_{\pi_i}}$, **go to Learner**
  Learner:
    If status = SAT
      If $||\mu_E - \mu_{\pi_i}||_2 \leq \epsilon$, **then return** π_i
        ▷ Converge. π_i is ϵ-close to π_E
      Add π_i to Π_S, $inf \leftarrow k, k \leftarrow sup$ ▷ Update Π_S, inf and reset k
    If status = UNSAT
      If $|k - inf| \leq \sigma$, **then return** $\pi^* \leftarrow \underset{\pi \in \Pi_S}{argmin} ||\mu_E - \mu_\pi||_2$
        ▷ Converge. k is too close to its lower bound.
      $k \leftarrow \alpha \cdot inf + (1 - \alpha)k$ ▷ Decrease k to learn for safety
    $\omega_{i+1} \leftarrow \underset{\omega}{argmax} \min_{\pi \in \Pi_S, \tilde{\pi} \in \Pi_S, cex \in CEX} \omega^T (k(\mu_E - \mu_\pi) + (1 - k)(\mu_{\tilde{\pi}} - \mu_{cex}))$
  ▷ Note that the multi-objective optimization function recovers AL when $k = 1$
    $\pi_{i+1}, \mu_{\pi_{i+1}} \leftarrow$ Compute the optimal policy π_{i+1} and its expected features $\mu_{\pi_{i+1}}$ for the MDP M with reward $R(s) = \omega_{i+1}^T f(s)$
  Go to next iteration

Algorithm 1 describes CEGAL in detail. With a constant $sup = 1$ and a variable $inf \in [0, sup]$ for the upper and lower bounds respectively, the learner

determines the value of k within $[inf, sup]$ in each iteration depending on the outcome of the verifier and uses k in solving (10) in line 27. Like most nonlinear optimization algorithms, this algorithm requires an initial guess, which is an initial safe policy π_0 to make Π_S nonempty. A good initial candidate would be the maximally safe policy for example obtained using PRISM-games [15]. Without loss of generality, we assume this policy satisfies Φ. Suppose in iteration i, an intermediate policy π_i learnt by the learner in iteration $i-1$ is verified to satisfy Φ, then we increase inf to $inf = k$ and reset k to $k = sup$ as shown in line 22. If π_i does not satisfy Φ, then we reduce k to $k = \alpha \cdot inf + (1-\alpha)k$ as shown in line 26 where $\alpha \in (0,1)$ is a step length parameter. If $|k - inf| \leq \sigma$ and π_i still does not satisfy Φ, the algorithm chooses from Π_S a best safe policy π^* which has the smallest margin to π_E as shown in line 24. If π_i satisfies Φ and is ϵ-close to π_E, the algorithm outputs π_i as show in line 19. For the occasions when π_i satisfies Φ and $inf = sup = k = 1$, solving (10) is equivalent to solving (1) as in the original AL algorithm.

Remark 2. The initial policy π_0 does not have to be maximally safe, although such a policy can be used to verify if Φ is satisfiable at all. Naively safe policies often suffice for obtaining a safe and performant output at the end. Such a policy can be obtained easily in many settings, e.g., in the grid-world example one safe policy is simply staying in the initial cell. In both cases, π_0 typically has very low performance since satisfying Φ is the only requirement for it.

Theorem 1. *Given an initial policy π_0 that satisfies Φ, Algorithm 1 is guaranteed to output a policy π^*, such that (1) π^* satisfies Φ, and (2) the performance of π^* is at least as good as that of π_0 when compared to π_E, i.e. $\|\mu_E - \mu_{\pi^*}\|_2 \leq \|\mu_E - \mu_{\pi_0}\|_2$.*

Proof sketch. The first part of the guarantee can be proven by case splitting. Algorithm 1 outputs π^* either when π^* satisfies Φ and is ϵ-close to π_E, or when $|k - inf| \leq \sigma$ in some iteration. In the first case, π^* clearly satisfies Φ. In the second case, π^* is selected from the set Π_S which contains all the policies that have been found to satisfy Φ so far, so π^* satisfies Φ. For the second part of the guarantee, the initial policy π_0 is the final output π^* if π_0 satisfies Φ and is ϵ-close to π_E. Otherwise, π_0 is added to Π_S if it satisfies Φ. During the iteration, if $|k - inf| \leq \sigma$ in some iteration, then the final output is $\pi^* = \underset{\pi \in \Pi_S}{argmin} ||\mu_E - \mu_\pi||_2$, so it must satisfy $\|\mu_E - \mu_{\pi^*}\|_2 \leq \|\mu_E - \mu_{\pi_0}\|_2$. If a learnt policy π^* satisfies Φ and is ϵ-close to π_E, then Algorithm 1 outputs π^* without adding it to Π_S. Obviously $\|\mu_E - \mu_\pi\|_2 > \epsilon, \forall \pi \in \Pi_S$, so $\|\mu_E - \mu_{\pi^*}\|_2 \leq \|\mu_E - \mu_{\pi_0}\|_2$.

Discussion. In the worst case, CEGAL will return the initial safe policy. However, this can be because a policy that simultaneously satisfies Φ and is ϵ-close to the expert's demonstrations does not exist. Comparing to AL which offers no safety guarantee and finding the maximally safe policy which has very poor performance, CEGAL provides a principled way of guaranteeing safety while retaining performance.

Convergence. Algorithm 1 is guaranteed to converge. Let inf_t be the t^{th} assigned value of inf. After inf_t is given, k is decreased from $k_0 = sup$ iteratively by

$k_i = \alpha \cdot inf_t + (1-\alpha)k_{i-1}$ until either $|k_i - inf_t| \leq \sigma$ in line 24 or a new safe policy is found in line 18. The update of k satisfies the following equality.

$$\frac{|k_{i+1} - inf_t|}{|k_i - inf_t|} = \frac{\alpha \cdot inf_t + (1-\alpha)k_i - inf_t}{k_i - inf_t} = 1 - \alpha \tag{11}$$

Thus, it takes no more than $1+\log_{1-\alpha} \frac{\sigma}{sup-inf_t}$ iterations for either the algorithm to converge in line 24 or a new safe policy to be found in line 18. If a new safe policy is found in line 18, inf will be assigned in line 22 by the current value of k as $inf_{t+1} = k$ which obviously satisfies $inf_{t+1} - inf_t \geq (1-\alpha)\sigma$. After the assignment of inf_{t+1}, the iterative update of k resumes. Since $sup - inf_t \leq 1$, the following inequality holds.

$$\frac{|inf_{t+1} - sup|}{|inf_t - sup|} \leq \frac{sup - inf_t - (1-\alpha)\sigma}{sup - inf_t} \leq 1 - (1-\alpha)\sigma \tag{12}$$

Obviously, starting from an initial $inf = inf_0 < sup$, with the alternating update of inf and k, inf will keep getting close to sup unless the algorithm converges as in line 24 or a safe policy ϵ-close to π_E is found as in line 19. The extreme case is that finally $inf = sup$ after no more than $\frac{sup-inf_0}{(1-\alpha)\sigma}$ updates on inf. Then, the problem becomes AL. Therefore, the worst case of this algorithm can have two phases. In the first phase, inf increases from $inf = 0$ to $inf = sup$. Between each two consecutive updates $(t, t+1)$ on inf, there are no more than $\log_{1-\alpha} \frac{(1-\alpha)\sigma}{sup-inf_t}$ updates on k before inf is increased from inf_t to inf_{t+1}. Overall, this phase takes no more than

$$\sum_{0 \leq i < \frac{sup-inf_0}{(1-\alpha)\sigma}} \log_{1-\alpha} \frac{(1-\alpha)\sigma}{sup - inf_0 - i \cdot (1-\alpha)\sigma} = \sum_{0 \leq i < \frac{1}{(1-\alpha)\sigma}} \log_{1-\alpha} \frac{(1-\alpha)\sigma}{1 - i \cdot (1-\alpha)\sigma} \tag{13}$$

iterations to reduce the multi-objective optimization problem to original apprenticeship learning and then the second phase begins. Since $k = sup$, the iteration will stop immediately when an unsafe policy is learnt as in line 24. This phase will not take more iterations than original AL algorithm does to converge and the convergence result of AL is given in [1].

In each iteration, the algorithm first solves a second-order cone programming (SOCP) problem (10) to learn a policy. SOCP problems can be solved in polynomial time by interior-point (IP) methods [12]. PCTL model checking for DTMCs can be solved in time linear in the size of the formula and polynomial in the size of the state space [7]. Counterexample generation can be done either by enumerating paths using the k-shortest path algorithm or determining a critical subsystem using either a SMT formulation or mixed integer linear programming (MILP) [23]. For the k-shortest path-based algorithm, it can be computationally expensive sometimes to enumerate a large amount of paths (i.e. a large k) when p^* is large. This can be alleviated by using a smaller p^* during calculation, which is equivalent to considering only paths that have high probabilities.

6 Experiments

We evaluate our algorithm on three case studies: (1) grid-world, (2) cart-pole, and (3) mountain-car. The cart-pole environment[1] and the mountain-car environment[2] are obtained from OpenAI Gym. All experiments are carried out on a quad-core i7-7700K processor running at 3.6 GHz with 16 GB of memory. Our prototype tool was implemented in Python[3]. The parameters are $\gamma = 0.99, \epsilon = 10, \sigma = 10^{-5}, \alpha = 0.5$ and the maximum number of iterations is 50. For the OpenAI-gym experiments, in each step, the agent sends an action to the OpenAI environment and the environment returns an observation and a reward (0 or 1). We show that our algorithm can guarantee safety while retaining the performance of the learnt policy compared with using AL alone.

6.1 Grid World

We first evaluate the scalability of our tool using the grid-world example. Table 1 shows the average runtime (per iteration) for the individual components of our tool as the size of the grid-world increases. The first and second columns indicate the size of the grid world and the resulting state space. The third column shows the average runtime that policy iteration takes to compute an optimal policy π for a known reward function. The forth column indicates the average runtime that policy iteration takes to compute the expected features μ for a known policy. The fifth column indicates the average runtime of verifying the PCTL formula using PRISM. The last column indicates the average runtime that generating a counterexample using COMICS.

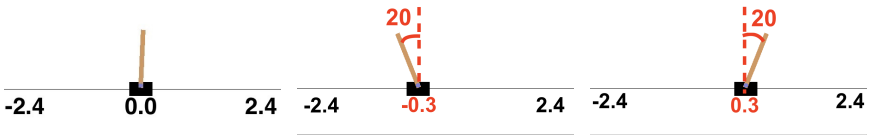
Table 1: Average runtime per iteration in seconds.

Size	Num. of States	Compute π	Compute μ	MC	Cex
8×8	64	0.02	0.02	1.39	0.014
16×16	256	0.05	0.05	1.43	0.014
32×32	1024	0.07	0.08	3.12	0.035
64×64	4096	6.52	25.88	22.877	1.59

6.2 Cart-Pole from OpenAI Gym

In the cart-pole environment as shown in Fig. 4(a), the goal is to keep the pole on a cart from falling over as long as possible by moving the cart either to the left or to the right in each time step. The maximum step length is $t = 200$. The position, velocity and angle of the cart and the pole are continuous values and observable, but the actual dynamics of the system are unknown[4].

[1] https://github.com/openai/gym/wiki/CartPole-v0
[2] https://github.com/openai/gym/wiki/MountainCar-v0
[3] https://github.com/zwc662/CAV2018
[4] The MDP is built from sampled data. The feature vector in each state contains 30 radial basis functions which depend on the squared Euclidean distances between current state and other 30 states which are uniformly distributed in the state space.



(a) (b) (c)

Fig. 4: (a) The cart-pole environment. (b) The cart is at -0.3 and pole angle is $-20°$. (c) The cart is at 0.3 and pole angle is $20°$.

A maneuver is deemed *unsafe* if the pole angle is larger than $\pm 20°$ while the cart's horizontal position is more than ± 0.3 as shown in Fig. 4(b) and 4(c). We formalize the safety requirement in PCTL as (14).

$$\Phi ::= P_{\leq p^*}[true\ \mathbf{U}^{\leq t}\ (angle \leq -20° \wedge position \leq -0.3) \vee (angle \geq 20° \wedge position \geq 0.3)] \tag{14}$$

Table 2: In the cart-pole environment, *higher* average steps mean better performance. The safest policy is synthesized using PRISM-games.

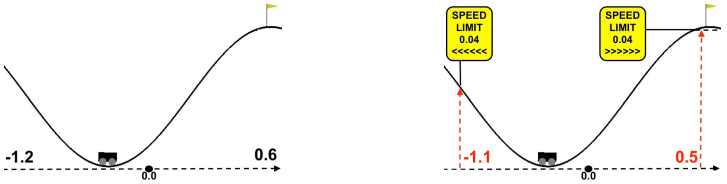
	MC Result	Avg. Steps	Num. of Iters
AL	49.1%	165	2
Safest Policy	0.0%	8	N.A.
$p^* = 30\%$	17.2%	121	10
$p^* = 25\%$	9.3%	136	14
$p^* = 20\%$	17.2%	122	10
$p^* = 15\%$	6.9%	118	22
$p^* = 10\%$	7.2%	136	22
$p^* = 5\%$	0.04%	83	50

We used 2000 demonstrations for which the pole is held upright without violating any of the safety conditions for all 200 steps in each demonstration. The safest policy synthesized by PRISM-games is used as the initial safe policy. We also compare the different policies learned by CEGAL for different safety threshold p^*s. In Table 2, the policies are compared in terms of model checking results ('MC Result') on the PCTL property in (14) using the constructed MDP, the average steps ('Avg. Steps') that a policy (executed in the OpenAI environment) can hold across 5000 rounds (the higher the better), and the number of iterations ('Num. of Iters) it takes for the algorithm to terminate (either converge to an ϵ-close policy, or converge due to σ, or terminate after 50 iterations). The policy in the first row is the result of using AL alone, which has the best performance but also a 49.1% probability of violating the safety requirement. The safest policy as shown in the second row is always safe has almost no performance at all. This policy simply letts the pole fall and thus does not risk moving the cart out of the range [-0.3, 0.3]. On the other hand, it is clear that the policies learnt using CEGAL always satisfy the safety requirement. From $p^* = 30\%$ to 10%, the performance of the learnt policy is comparable to that of the AL policy.

However, when the safety threshold becomes very low, e.g., $p^* = 5\%$, the performance of the learnt policy drops significantly. This reflects the phenomenon that the tighter the safety condition is the less room for the agent to maneuver to achieve a good performance.

6.3 Mountain-Car from OpenAI Gym

Our third experiment uses the mountain-car environment from OpenAI Gym. As shown in Fig. 5(a), a car starts from the bottom of the valley and tries to reach the mountaintop on the right as quickly as possible. In each time step the car can perform one of the three actions, accelerating to the left, coasting, and accelerating to the right. The agent fails if the step length reaches the maximum ($t = 66$). The velocity and position of the car are continuous values and observable while the exact dynamics are unknown[5]. In this game setting, the car cannot reach the right mountaintop by simply accelerating to the right. It has to accumulate momentum first by moving back and forth in the valley. The safety rules we enforce are shown in Fig. 5(b). They correspond to speed limits when the car is close to the left mountaintop or to the right mountaintop (in case it is a cliff on the other side of the mountaintop). Similar to the previous experiments, we considered 2000 expert demonstrations for which all of them successfully reach the right mountaintop without violating any of the safety conditions. The average number of steps for the expert to drive the car to the right mountaintop is 40. We formalize the safety requirement in PCTL as (15).



(a) (b)

Fig. 5: (a) The original mountain-car environment. (b) The mountain-car environment with traffic rules: when the distance from the car to the left edge or the right edge is shorter than 0.1, the speed of the car should be lower than 0.04.

$$\Phi ::= P_{\leq p^*}[true\ \mathbf{U}^{\leq t}\ (speed \leq -0.04 \wedge position \leq -1.1) \vee (speed \geq 0.04 \wedge position \geq 0.5)] \tag{15}$$

We compare the different policies using the same set of categories as in the cart-pole example. The numbers are averaged over 5000 runs. As shown in the first row, the policy learnt via AL[6] has the highest probability of going over

[5] The MDP is built from sampled data. The feature vector for each state contains 2 exponential functions and 18 radial basis functions which respectively depend on the squared Euclidean distances between the current state and other 18 states which are uniformly distributed in the state space.

[6] AL did not converge to an ϵ-close policy in 50 iterations in this case.

the speed limits. We observed that this policy made the car speed up all the way to the left mountaintop to maximize its potential energy. The safest policy corresponds to simply staying in the bottom of the valley. The policies learnt via CEGAL for safety threshold p^* ranging from 60% to 50% not only have lower probability of violating the speed limits but also achieve comparable performance. As the safety threshold p^* decreases further, the agent becomes more conservative and it takes more time for the car to finish the task. For $p^* = 20\%$, the agent never succeeds in reaching the top within 66 steps.

Table 3: In the mountain-car environment, *lower* average steps mean better performance. The safest policy is synthesized via PRISM-games.

	MC Result	Avg. Steps	Num. of Iters
Policy Learnt via AL	69.2%	54	50
Safest Policy	0.0%	*Fail*	N.A.
$p^* = 60\%$	43.4%	57	9
$p^* = 50\%$	47.2%	55	17
$p^* = 40\%$	29.3%	61	26
$p^* = 30\%$	18.9%	64	17
$p^* = 20\%$	4.9%	*Fail*	40

7 Related Work

A taxonomy of AI safety problems is given in [3] where the issues of misspecified objective or reward and insufficient or poorly curated training data are highlighted. There have been several attempts to address these issues from different angles. The problem of *safe exploration* is studied in [17] and [8]. In particular, the latter work proposes to add a safety constraint, which is evaluated by amount of damage, to the optimization problem so that the optimal policy can maximize the return without violating the limit on the expected damage. An obvious shortcoming of this approach is that actual failures will have to occur to properly assess damage.

Formal methods have been applied to the problem of AI safety. In [5], the authors propose to combine machine learning and reachability analysis for dynamical models to achieve high performance and guarantee safety. In this work, we focus on probabilistic models which are natural in many modern machine learning methods. In [20], the authors propose to use formal specification to synthesize a control policy for reinforcement learning. They consider formal specifications captured in Linear Temporal Logic, whereas we consider PCTL which matches better with the underlying probabilistic model. Recently, the problem of *safe reinforcement learning* was explored in [2] where a monitor (called shield) is used to enforce temporal logic properties either during the learning phase or execution phase of the reinforcement learning algorithm. The shield provides a list of safe actions each time the agent makes a decision so that the temporal property is preserved. In [11], the authors also propose an approach for controller synthesis in reinforcement learning. In this case, an SMT-solver is used to find a scheduler (policy) for the synchronous product of an MDP and a DTMC

so that it satisfies both a probabilistic reachability property and an expected cost property. Another approach that leverages PCTL model checking is proposed in [16]. A so-called abstract Markov decision process (AMDP) model of the environment is first built and PCTL model checking is then used to check the satisfiability of safety specification. Our work is similar to these in spirit in the application of formal methods. However, while the concept of AL is closely related to reinforcement learning, an agent in the AL paradigm needs to learn a policy from demonstrations without knowing the reward function a priori.

A distinguishing characteristic of our method is the tight integration of formal verification with learning from data (apprenticeship learning in particular). Among imitation or apprenticeship learning methods, margin based algorithms [1], [18], [19] try to maximize the margin between the expert's policy and all learnt policies until the one with the smallest margin is produced. The apprenticeship learning algorithm proposed by Abbeel and Ng [1] was largely motivated by the support vector machine (SVM) in that features of expert demonstration is maximally separately from all features of all other candidate policies. Our algorithm makes use of this observation when using counterexamples to steer the policy search process. Recently, the idea of learning from failed demonstrations started to emerge. In [21], the authors propose an IRL algorithm that can learn from both successful and failed demonstrations. It is done by reformulating maximum entropy algorithm in [24] to find a policy that maximally deviates from the failed demonstrations while approaching the successful ones as much as possible. However, this entropy-based method requires obtaining many failed demonstrations and can be very costly in practice.

Finally, our approach is inspired by the work on formal inductive synthesis [10] and counterexample-guided inductive synthesis (CEGIS) [22]. These frameworks typically combine a constraint-based synthesizer with a verification oracle. In each iteration, the agent refines her hypothesis (i.e. generates a new candidate solution) based on counterexamples provided by the oracle. Our approach can be viewed as an extension of CEGIS where the objective is not just functional correctness but also meeting certain learning criteria.

8 Conclusion and Future work

We propose a counterexample-guided approach for combining probabilistic model checking with apprenticeship learning to ensure safety of the apprenticehsip learning outcome. Our approach makes novel use of counterexamples to steer the policy search process by reformulating the feature matching problem into a multi-objective optimization problem that additionally takes safety into account. Our experiments indicate that the proposed approach can guarantee safety and retain performance for a set of benchmarks including examples drawn from OpenAI Gym. In the future, we would like to explore other imitation or apprenticeship learning algorithms and extend our techniques to those settings.

Acknowledgement. This work is funded in part by the DARPA BRASS program under agreement number FA8750-16-C-0043 and NSF grant CCF-1646497.

References

1. P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 1–, New York, NY, USA, 2004. ACM.
2. M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. Safe reinforcement learning via shielding. *CoRR*, abs/1708.08611, 2017.
3. D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in AI safety. *CoRR*, abs/1606.06565, 2016.
4. R. Bellman. A markovian decision process. 6:15, 04 1957.
5. J. H. Gillulay and C. J. Tomlin. Guaranteed safe online learning of a bounded system. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 2979–2984. IEEE, 2011.
6. T. Han, J. P. Katoen, and D. Berteun. Counterexample generation in probabilistic model checking. *IEEE Transactions on Software Engineering*, 35(2):241–257, March 2009.
7. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, Sep 1994.
8. D. Held, Z. McCarthy, M. Zhang, F. Shentu, and P. Abbeel. Probabilistically safe policy transfer. *CoRR*, abs/1705.05394, 2017.
9. N. Jansen, E. Ábrahám, M. Scheffler, M. Volk, A. Vorpahl, R. Wimmer, J. Katoen, and B. Becker. The COMICS tool - computing minimal counterexamples for discrete-time markov chains. *CoRR*, abs/1206.0603, 2012.
10. S. Jha and S. A. Seshia. A theory of formal synthesis via inductive learning. *Acta Informatica*, 54(7):693–726, Nov 2017.
11. S. Junges, N. Jansen, C. Dehnert, U. Topcu, and J.-P. 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.
12. Y.-J. Kuo and H. D. Mittelmann. Interior point methods for second-order cone programming and or applications. *Computational Optimization and Applications*, 28(3):255–285, 2004.
13. M. Kwiatkowska, G. Norman, and D. Parker. Prism: Probabilistic symbolic model checker. *Computer Performance Evaluation/TOOLS*, 2324:200–204, 2002.
14. M. Kwiatkowska and D. Parker. *Automated Verification and Strategy Synthesis for Probabilistic Systems*, pages 5–22. Springer International Publishing, Cham, 2013.
15. M. Kwiatkowska, D. Parker, and C. Wiltsche. Prism-games: verification and strategy synthesis for stochastic multi-player games with multiple objectives. *International Journal on Software Tools for Technology Transfer*, Nov 2017.
16. G. R. Mason, R. C. Calinescu, D. Kudenko, and A. Banks. Assured reinforcement learning for safety-critical applications. In *Doctoral Consortium at the 10th International Conference on Agents and Artificial Intelligence*. SciTePress, 2017.
17. T. M. Moldovan and P. Abbeel. Safe exploration in markov decision processes. *arXiv preprint arXiv:1205.4810*, 2012.
18. A. Y. Ng and S. J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
19. N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich. Maximum margin planning. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 729–736, New York, NY, USA, 2006. ACM.

20. D. Sadigh, E. S. Kim, S. Coogan, S. S. Sastry, and S. A. Seshia. A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications. *CoRR*, abs/1409.5486, 2014.
21. K. Shiarlis, J. Messias, and S. Whiteson. Inverse reinforcement learning from failure. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1060–1068. International Foundation for Autonomous Agents and Multiagent Systems, 2016.
22. A. Solar-Lezama, L. Tancau, R. Bodik, S. Seshia, and V. Saraswat. Combinatorial sketching for finite programs. *SIGOPS Oper. Syst. Rev.*, 40(5):404–415, oct 2006.
23. R. Wimmer, N. Jansen, E. Ábrahám, B. Becker, and J.-P. Katoen. *Minimal Critical Subsystems for Discrete-Time Markov Models*, pages 299–314. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
24. B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, pages 1433–1438. AAAI Press, 2008.