Risk-Averse Control of Markov Decision Processes with ω-regular Objectives

Ruediger Ehlers[1], Salar Moarref[2], and Ufuk Topcu[3]

Abstract— **Many control problems in environments that can be modeled as Markov decision processes (MDPs) concern infinite-time horizon specifications. The classical aim in this context is to compute a control policy that maximizes the probability of satisfying the specification. In many scenarios, there is however a non-zero probability of failure in every step of the system's execution. For infinite-time horizon specifications, this implies that the specification is violated with probability 1 in the long run no matter what policy is chosen, which prevents previous policy computation methods from being useful in these scenarios.**

In this paper, we introduce a new optimization criterion for MDP policies that captures the task of working towards the satisfaction of some infinite-time horizon ω-regular specification. The new criterion is applicable to MDPs in which the violation of the specification cannot be avoided in the long run. We give an algorithm to compute policies that are optimal in this criterion and show that it captures the ideas of *optimism* and *risk-averseness* in MDP control: while the computed policies are optimistic in that a MDP run enters a failure state relatively late, they are risk-averse by always maximizing the probability to reach their respective next goal state. We give results on two robot control scenarios to validate the usability of risk-averse MDP policies.

I. Introduction

The class of ω-regular specifications allows to concisely capture long-term tasks for systems to be controlled. Consequently, they have not only been used as specification formalism for the control of deterministic systems, but also found applications in control of probabilistic systems. In the probabilistic case, the objective is typically to ensure that the specification holds almost surely or with the highest possible probability.

There are however many systems that do not admit any control strategy that satisfies an ω-regular objective with a non-zero probability. In such a case, all controllers are equally bad: they violate the specification almost surely (or surely). If, for example, we have a robot control scenario where there is always a small probability that the robot moves towards a wall (due to external influences), then a specification that forbids colliding with the wall cannot be fulfilled with a non-zero probability, as colliding with a wall almost surely eventually happens. Yet, researchers have proposed many approaches for controlling robots in such environments in practice. In a nutshell, these approaches are *optimistic*: why should we be intimidated by events that are unavoidable but occur with small probability even in long time spans when we can still satisfy the specification for some time? Such approaches are typically also *risk-averse*: within the actions that are available to the robot, those that avoid the violation of the specification as long as possible are preferred. A reasonable strategy for the robot could, for example, try to stay clear of the walls and immediately take action when it happens to get closer to the wall at runtime. In this way, the robot could work towards satisfying its goals even though in the long run, it will eventually collide with a wall almost surely.

On a theoretical level, the infinite-horizon nature of ω-regular specifications however prevents the immediate application of *optimism*. If, with probability 1, the specification is violated no matter what policy is used for controlling the system, then all control policies are equally bad and no best policy can be generated. While this fact may advocate for an approach to system control that is not based on ω-regular objectives, the infinitary nature of such objectives allows to abstract from many details of the specification. As an example, we can state in the ω-regular setting that the robot should visit each of two regions in a workspace infinitely often, which is a concise representation of the task of patrolling between these regions. The specification does not impose maximal times between visits to the regions, which allows to optimize the risk-averseness of the policy. Deviating from the concept of ω-regular objectives would mean to impose time bounds between the visits to the regions. But then we get a tradeoff between optimizing for satisfying the specification as long as possible and the lengths of the patrolling periods. So it is desirable to keep the simplicity and conciseness of ω-regular specifications to allow optimizing the probability to satisfy the specification for at least some time.

In this paper, we show how to compute *optimistic*, yet *risk-averse* policies for satisfying ω-regular objectives in Markov decision processes (MDPs). We define an optimization criterion that captures the task of computing policies that satisfy ω-regular control objectives as long as possible, and give an algorithm to compute these policies. The basic idea is that we require the policy to have a labeling that describes which states are considered to be *goal states* by the policy, i.e., for which visiting them infinitely often ensures that the specification is satisfied. An *optimally risk-averse policy* maximizes the probability of reaching the next goal state from the respective previous goal state. We argue that this criterion matches the intuitive idea that the controller should satisfy the specification as long as possible even if its

[1] Rüdiger Ehlers is with the University of Bremen and DFKI GmbH, Bremen, Germany

[2] Salar Moarref is with the University of Pennsylvania, Philadelphia, PA, USA.

[3] Ufuk Topcu is with the University of Austin at Texas, TX, USA.

violation is almost surely unavoidable in the long run. We validate the usability of our risk-averse policy definition and the scalability of our policy computation algorithm on two case studies for robot control in probabilistic environments.

II. Related Work

MDPs are widely used in many areas such as engineering, economics and biology, and have been successfully used to model and control autonomous robots with uncertainty in their sensing and actuation (see e.g., [1], [2], [3], [4]). In these domains, the behavior of the system cannot be predicted with certainty, but it can be modeled probabilistically through simulations or empirical trials. Our results in this paper can be used in *practical* settings in which the system cannot be controlled to satisfy a specification in the long run, but some amount of risk-taking is acceptable. This is also in contrast to earlier works on computing risk-averse policies for MDPs without control objective (see, e.g., [5]), which are concerned with reducing the risk of obtaining low *rewards* in an MDP run at the expense of lowering the expected rewards.

MDPs are also referred to as $1\frac{1}{2}$-player games and belong to a broader class of *stochastic games*. The algorithmic study of stochastic games with respect to ω-regular objectives has recently attracted significant attention [6], [7], [8], [9], [10], [11]. See [12] for a detailed survey. The central question about a game is whether a player has a strategy for winning the game. There are several definitions for *winning* in a stochastic game [12]. For example, one may ask if a player has a strategy that ensures a winning outcome of the game, no matter how the other player chooses her actions (*sure winning*), or one may ask if a player has a strategy that achieves a winning outcome with probability 1 (*almost-sure winning*). In contrast to these *qualitative* winning criteria, the *quantitative* solution [13], [9] amounts to computing the value of the game, i.e., the maximal probability of winning that a player can guarantee against any strategy chosen by the opponent. The choice of MDPs in this paper is motivated by their manageable complexity compared to more general classes of stochastic games, and by their applicability to many control problems. Ding et al. [14] gave an approach to compute MDP policies that maximize the probability of satisfying an ω-regular specification. They applied their algorithm to robot indoor navigation. Svorenova et al. [15] considered the problem of minimizing the expected cost in between reaching *goal states* in MDPs for ω-regular specifications. Our work uses a similar notion of goal states. None of the mentioned works consider the synthesis of risk-averse policies in case there is no strategy that wins with a probability of greater than 0.

III. Preliminaries

a) MDPs: A *Markov decision process* is defined as a tuple $\mathcal{M} = (S, A, \Sigma, P, L, s_0)$, where S is a finite set of *states*, A is a finite set of *actions*, Σ is the *label alphabet*, $P : S \times A \to \mathcal{P}(S) \cup \{\perp\}$ is the *transition function*, where $\mathcal{P}(S)$ denotes the probability distributions over S and $\perp$ denotes a disallowed action in a state, $L : S \to \Sigma$ is the labeling function of $\mathcal{M}$, and $s_0 \in S$ is the initial state of the Markov chain. We say that some finite sequence $\pi = \pi_0 \ldots \pi_n \in S^*$ is a *finite trace* (or *run*) of $\mathcal{M}$ if there exists a sequence of actions $\rho = \rho_0 \ldots \rho_{n-1} \in A^*$ such that for all $i \in \{0, \ldots, n-1\}$, we have $P(\pi_i, \rho_i) \neq \perp$ and $P(\pi_i, \rho_i)(\pi_{i+1}) > 0$. We say that the combined probability of (π, ρ) is $\prod_{i=0}^{n-1} P(\pi_i, \rho_i)(\pi_{i+1})$. The definition of finite traces carries over to infinite traces.

A *Markov chain* is a Markov decision process (MDP) in which $A = \{\cdot\}$. A Markov chain introduces the usual probability measure over sets of infinite traces.

A *policy* for an MDP is a function $f : S^* \to \mathcal{P}(A)$ such that for all $s_0 \ldots s_n \in S^*$, we have $f(s_0 \ldots s_n)(a) = 0$ for all actions a such that $P(s_n, a) = \perp$. A policy induces an infinite-state Markov chain $\mathcal{C}' = (S', \{\cdot\}, \Sigma, P', L', s_0)$ with $S' = S^*$, $L'(s_0 \ldots s_n) = L(s_n)$ for all $s_0 \ldots s_n \in S'$, and for all $s_0 \ldots s_n, u_0 \ldots u_m \in S'$, we have $P'(s_0 \ldots s_n, \cdot)(u_0 \ldots u_m) = \sum_{a \in A} P(s_n, a)(u_m) \cdot f(s_0 \ldots s_n)(a)$ if $u_0 \ldots u_{m-1} = s_0 \ldots s_n$, and $P'(s_0 \ldots s_n, \cdot)(u_0 \ldots u_m) = 0$ otherwise.

Policies for MDPs can be *positional* or *finite-state*. For a positional policy, for all $t = t_0 \ldots t_n \in S^*$ and $t' = t'_0 \ldots t'_m \in S^*$, we have that $f(t) = f(t')$ if $t_n = t'_m$. For a finite-state policy, there exists a finite-state automaton $\mathcal{F} = (Q, S, \delta, q_0)$ with a finite set of states Q, $q_0 \in Q$, and $\delta : Q \times S \to Q$ such that there is a function $f' : Q \to \mathcal{P}(A)$ such that for all $t = t_0 \ldots t_n \in S^*$, we have that $f(t) = f'(q)$ for $q = \delta(\ldots \delta(\delta(q_0, t_0), t_1), \ldots, t_n)$.

In literature, MDPs often also have a *reward function*. As in some other work on ω-regular MDP control (see, e.g., [14]), we do not need it in this paper and have thus omitted the reward function in the MDP definition. An MDP can be represented graphically by drawing the states as nodes in a graph, marking the initial state and letting the transitions be represented by groups of *edges*, which are in turn labeled by their transition probabilities. The groups of edges are labeled by their actions. Disallowed actions, i.e., for which we have $P(s, a) = \perp$, are not shown.

b) Parity automata and ω-regular specifications: Given an alphabet Σ, an ω-regular specification is a subset of Σ^ω that is representable as the language of a *deterministic parity word automaton*. These automata are defined as tuples $\mathcal{A} = (Q, \Sigma, \delta, q_0, C)$, where Q is a finite set of states, Σ is an alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function of $\mathcal{A}$, $q_0 \in Q$ is the initial state of the automaton, and $C : Q \to \mathbb{N}$ is the *coloring function*. Given a word $w = w_0 w_1 \ldots \in \Sigma^\omega$, $\mathcal{A}$ induces a *trace* $t = t_0 t_1 \ldots \in Q^\omega$ such that for all $i \in \mathbb{N}$, we have $t_{i+1} = \delta(t_i, w_i)$. Let inf be a function that maps an infinite sequence onto the elements of the sequence that occur infinitely often in it. A trace t of $\mathcal{A}$ is called *accepting* if $\max(\mathsf{inf}(c(t_0)c(t_1)c(t_2)\ldots))$ is even. An automaton is said to accept a word w if there exists an accepting trace for it. The set of all words accepted by the automaton is called its *language*.

c) Reachability MDPs: A reachability MDP $\mathcal{M} = (S, A, \Sigma, P, L, s_0, g)$ consists of the usual MDP elements plus a function $g : S \to \{0,1\}$, which assigns to every state $s \in S$ either 0 or 1 depending on whether it is a *goal state* or not. A policy f for $\mathcal{M}$ induces for every state s a *value* $v(s) \in [0,1]$ that denotes the probability measure of the traces starting in s and eventually visiting a state $s' \in S$ with $g(s') = 1$ when executing the policy, i.e., in the Markov chain induced by $\mathcal{M}$ and f starting from state s. A policy that maximizes the values from all starting states is called *optimal* and it is known that in reachability MDPs, positional optimal policies exist [6]. The values of the states induced by an optimal policy are also called the *state values* of the reachability MDP. These can be computed either by *policy iteration* or *value iteration* algorithms [16]. In the latter case, a sequence of vectors $\vec{x}_1, \vec{x}_2, \ldots \in [0,1]^{|S|}$ is computed such that for every $i \in \mathbb{N}$, x_{i+1} is closer to the vector of state values than x_i. Value iteration is normally programmed to abort computation if at some point, $||x_{i+1} - x_i|| \leq \epsilon$ for some value ϵ and some norm $||\cdot||$. When starting with $\vec{x}_0$ being equivalent to g, the approximations are all under-approximations of the actual state values (modulo rounding errors).

IV. Problem Definition

Definition 1 (Parity MDP): The product of an MDP $\mathcal{M} = (S, A, \Sigma, P, L, s_0)$ and a parity word automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, C)$ is an MDP $\mathcal{M}' = (S', A, \Sigma, P', C', s'_0)$ with a coloring function instead of a labeling function where:

$$\begin{aligned} S' &= S \times Q, \\ s'_0 &= (s_0, q_0), \\ C'(s,q) &= C(q) \text{ for all } (s,q) \in S', \text{and} \\ P'((s,q),a)((s',q')) &= \begin{cases} P'(s,a)(s') & \text{if } q' = \delta(q, L(s')) \\ 0 & \text{else} \end{cases} \\ &\quad \text{for all } (s,q),(s',q') \in S', a \in A. \end{aligned}$$

An infinite trace $t_0 t_1 \ldots \in S'^{\omega}$ in $\mathcal{M}'$ is said to be *accepting* if the highest number occurring infinitely often in the sequence $C'(t_0)C'(t_1)\ldots$ is even.

A parity MDP captures a control problem in a probabilistic environment. Let us consider an example.

Example 1: As a first example, we consider a simple robot with unicycle dynamics in a two-dimensional gridded world. The workspace, which we depict in Figure 1, has 70×40 cells and the robot always has one out of eight possible current directions. The speed of the robot is constant, and it needs to avoid hitting the workspace boundaries or the static obstacles. In order to model the scenario as an MDP, we use a semantics with a fixed time step. In every time step, we shift the current cell into the current direction of travel by 2 cells, extend the resulting rectangle by 0.1 into every direction to account for imprecise motion, and then assign transition probabilities that are proportional to the overlap of the rectangle with the world cells. There is an additional special error state in the MDP that represents crashes. In every step, the policy can decide to increase or decrease the current direction by 1 tick (out of 8). This

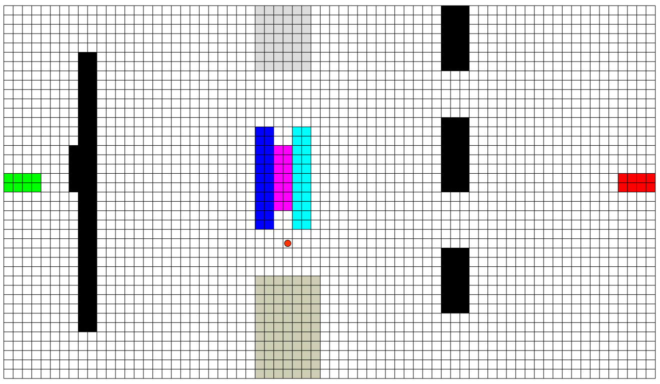
Fig. 1. Workspace for the single-robot example.

turning operation may fail with a probability of 0.2, and in the case of failure, the direction of the robot is not changed. The MDP has $70 \cdot 40 \cdot 8 + 1 = 22401$ states, 67201 state/action pairs, and 681591 *edges*, i.e., pairs (s, a, s') in the MDP $\mathcal{M} = (S, A, \Sigma, P, L, s_0)$ with $P(s,a)(s') > 0$.

The specification for the robot is represented as a 15 state parity automaton. It encodes four requirements:

- The left-most marked part of the workspace should be visited infinitely often,
- the right-most marked part of the workspace should be visited infinitely often,
- either the top marked part of the workspace must be visited only finitely often, or the bottom one, or both, and
- infinitely often, the regions in the middle shall be visited strictly in the middle-left-right order.

The product MDP of the MDP and the parity automaton has 366015 states, out of which 2196 are unreachable (and can be removed).

A classical problem over MDPs with ω-regular optimization criteria is to find a policy that maximizes the probability that a trace is accepting. In the product MDP from Example 1, there is however no policy that raises this probability above 0. This follows from the fact that no matter what the policy does, with a probability of at least 0.2, the robot continues to travel into the current direction. By the limited size of the workspace, colliding with the workspace boundaries takes at most 35 steps, and thus, a very conservative lower bound on the probability for a crash within 35 steps is $(0.2)^{35}$ at *every* step of the MDPs execution. In the long run, the collision is thus unavoidable with probability 1.

Despite the fact that the parity MDP does not admit a good policy in the traditional sense, we may want to compute a policy that works towards the satisfaction of the specification as long as possible while avoiding unnecessary *risks*. We formalize this objective in the following definition:

Definition 2: Let $\mathcal{M} = (S, A, \Sigma, P, C, s_0)$ be a parity MDP. We say that some control policy $f : S^* \to A$ has a *risk-averseness probability* $p \in [0,1]$ if there exist labelings $l : S^* \to \mathbb{N}$ and $l' : S^* \to \mathbb{B}$ and a Markov chain $\mathcal{C}'$ induced by $\mathcal{M}$ and f with the following properties:

- There exists some number $k \in \mathbb{N}$ such that for all $t_0 t_1 t_2 \ldots \in S^\omega$, there are at most k many indices $i \in \mathbb{N}$ for which we have $l(t_0 \ldots t_i) > l(t_0 \ldots t_i t_{i+1})$.
- For all $t_0 t_1 \ldots t_n \in S^*$, we have that $l(t_0 \ldots t_n)$ is even, and $l'(t_0 \ldots t_n) = \mathbf{true}$ implies that $C(t_n) \geq l(t_0 \ldots t_n)$ and that $C(t_n)$ is even.
- For all $t_0 t_1 \ldots t_n \in S^*$, if $C(t_n)$ is odd, then $l(t_0 \ldots t_n) > C(t_n)$.
- For all $t = t_0 t_1 \ldots t_n \in S^*$ with either (a) $l'(t) = \mathbf{true}$ or (b) $t = s_0$, the probability measure in $\mathcal{C}'$ to reach some state $t\,t'_0 \ldots t'_m \in S^*$ with $l'(t\,t'_0 \ldots t'_m) = \mathbf{true}$ from state t is at least p.

The labelings l and l' in Definition 2 augment a policy with the information what *goal color* the policy is trying to reach and when a *goal* has been reached. A goal must always be even-colored, but along different traces, different goals are allowed. From every goal state, the next goal state must be reached with probability at least p. Together with the first three requirements in Definition 2, this implements the parity acceptance condition, as they together state that the goal color can only decrease finitely often along a trace. The parity acceptance condition does not need to be fulfilled with strictly positive probability in the long run, however, as in between two visits to goal states, the policy may fail with probability $(1-p)$. Thus, we only require the parity acceptance condition to hold on those paths on which goal states are reached infinitely often (which may have probability measure 0). The strategy can choose goal states in a way that maximizes the probability of reaching the respective next goal state. Thus, the higher the value of p is, the more averse to the risk to miss the next goal the control policy needs to be.

The reader may wonder why mentioning the labeling function l' is actually necessary in Definition 2, as one could simply implicitly set $l'(t_0 \ldots t_n) = \mathbf{true}$ whenever $C(t_n) \geq l(t_0 \ldots t_n)$ and $C(t_n)$ is even. However, this change often requires the policy to be able to reach the next goal from state t_n with probability p in the induced Markov chain without raising the target goal color $l(\cdot)$, which is not always possible in a p-risk-averse strategy. Figure 2 shows an example in which increasing the color of state q_1 to 2 (which is even) would then reduce the maximally implementable risk-averseness level from 0.68 to 0.64. As changing an odd color to an even one only makes the parity acceptance condition easier to satisfy, this is a very unintuitive property. To avoid it, we thus chose to make the labeling function l' explicit.

Using Definition 2, we can now state the main problem considered in this paper:

Definition 3 (Optimal risk-averse policy synthesis): Given a parity MDP, the optimal risk-averse policy synthesis is to find the highest value p such that a policy for the MDP with risk-averseness level p exists, and to find such a policy.

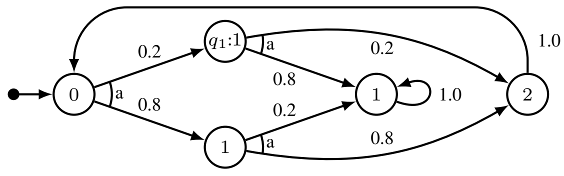

Fig. 2. An MDP in which for risk-averseness level $p = 0.68$, state q_1 is not winning, but the state is reachable on the (unique) p-risk-averse policy. All states are labeled by their colors.

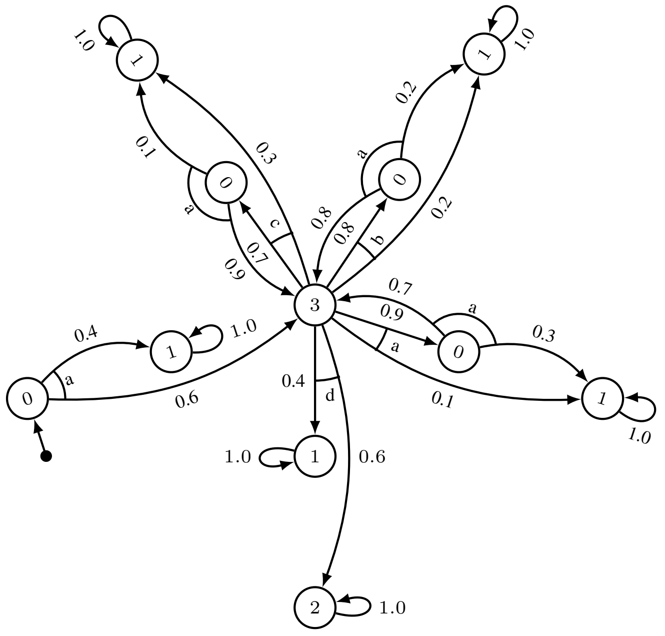

Fig. 3. An example parity MDP that admits a 0.54-risk-averse finite-memory policy, but no such positional policy. All states are labeled by their colors.

V. Computing Risk-Averse Policies

In this section, we describe an algorithm to compute risk-averse policies in parity MDPs. The algorithm produces finite-memory strategies that are not necessarily positional. This may appear to be a flaw of the algorithm, as memoryless policies suffice for maximizing the probability for a trace to satisfy a parity objective in MDPs [13]. However, optimal risk-averse strategies *do* require memory in general, which we show by means of an example.

Example 2: Figure 3 shows a parity MDP. It has four colors, and all states with color 1 are sink states, i.e., from which no possible goal state can be reached. The center state has the highest and odd color, so it may only be visited finitely often. Any policy cannot avoid either ending up in a sink state or visiting the middle state at least every second step, unless eventually action d is chosen by the policy. If the policy chooses action a in the initial state, and then immediately chooses d, it reaches the state with color 2 with a probability of $0.6 \cdot 0.6 = 0.36$. The resulting policy is thus 0.36-risk averse. However, there exists a better policy: when the state with color 3 is visited for the first time, action a should be taken, then action b, c, and finally action d. By declaring all color 0 states to be goal states, the resulting policy then has a risk averseness level of $\min(0.6 \cdot 0.9, 0.7 \cdot 0.8, 0.8 \cdot 0.7, 0.9 \cdot 0.6) = 0.54$. Thus, the best next action in the state with color 3 depends on the history of the trace. While this example only shows that memory is needed in optimally risk-averse policies, the fact that finite memory suffices follows from the correctness of our algorithm described below.

A. *p-risk-averse policy computation*

Let us assume that p is fixed and that we want to compute a p-risk-averse MDP control policy. The algorithm that we describe in this section computes the set of states from which a p-risk-averse policy exists. We call such states *winning*. The policies computed sometimes make use of non-winning states, which may be counter-intuitive at first. Figure 2 shows an example MDP where this is the case: from state q_1, the probability of reaching a next goal state is only 0.2, but the optimal 0.68-risk-averse policy from the initial state requires that even after reaching q_1, state q_2 is labeled as being a goal state if it is subsequently reached.

Whenever a goal state is reached, the only information about the history of the trace that may need to be retained is (1) how often the goal color may still be decreased before the limit of k is reached, and (2) what the current goal color is. This follows from the fact that the computation of probabilities is *reset* at goal states. Our algorithm makes use of this fact by planning policies from goal state to goal state(s). It iterates over all possible value combinations for the current goal color and the number of remaining goal color reductions.

Definition 4: We say that a state q is (k, c)-winning (for some fixed risk averseness level p) if there exists a p-risk-averse policy f from q as initial state with labels l and l' such that $l(\epsilon) = c$ and along all traces of the policy, goal colors are never decreased more than k times. We call such policies p-(k, c)-risk-averse.

Proposition 1: Some parity MDP admits a p-risk-averse policy if the initial state is (k, c)-winning for some values of $c \in \mathbb{N}$ and $k \in \mathbb{N}$.

Proof: Follows directly from Def. 2 and Def. 4. ■

This proposition allows us to frame the search for a p-risk-averse policy as an iterative process, which we base on the following lemma. Let $c_{maxEven}$ be the least *even* upper bound on the colors occurring in the parity MDP.

Lemma 1: A state q is winning for some values of (k, c) with $c \leq c_{maxEven}$ and even c if and only if there exists a policy such that, with probability p, eventually either:

- some even-colored state q' is visited that is winning for $(k - 1, 0)$, or
- some even-colored state q' with $C(q') = c'$ for $c' \geq c$ is visited that is winning for (k, c') while no odd color $\geq c'$ is visited along the way to q'.

Proof: We prove the claim by induction over (k, c) with even c. The order of induction that we use is lexicographic in $(k, -1 \cdot c)$.

Induction basis: For the case $(k, c) = (0, c_{maxEven})$, the only way for a state to be (k, c)-winning is for a policy from that state to exist such that, with probability at least p, a state is eventually visited that has color c and is (k, c)-winning again. This is exactly the only condition from the claim that is applicable in this case.

Induction step: ($\Rightarrow$) Let f' be a policy from q such that, on every trace of the policy, the goal colors decrease at most k times, and let $l(\epsilon) = c$ for the labelings (l, l') assigned to the policy. The probability to reach the next goal must be at least p in order for the state to be (k, c)-winning. A goal can either have a color greater than or equal to c or a color less than c. In the latter case, the goal state must be (k', c')-winning for some c' and some $k' < k$. As all such states are also $(k - 1, 0)$-winning (by definition), this case is covered by cases in the claim. If a goal with color c' is reached, then either no state with color $\geq c$ is visited along the way and $c' = c$, or alternatively $c' > c$ and no state with color $\geq c'$ is visited along the way. Both cases are covered by the case list in the claim.

($\Leftarrow$) Now let q be a state from which a policy to visit some goal state q' with probability p exists. State q' may be a (k, c)-winning state, but does not have to be one. If, on the way to q', a state with an odd color $c' > c$ is visited, this requires that the label function l' of the policy has to be greater than c' on the way from q to q'. So for the trace to count towards the probability mass of p, state q' needs to be either $(k, c' + x)$-winning (for even $x \geq 2$) or alternatively $(k - 1, c')$-winning. Since the set of $(k, c' + x)$-winning states is contained in the $(k, c' + 2)$-winning states and the $(k-1, c')$-winning states are a subset of the $(k-1, 0)$-winning states (by definition), we can assume, without loss of generality, that a $(k, c' + 2)$-winning or $(k - 1, 0)$-winning state is visited.

We construct the p-risk averse policy f with associated labels (l, l') that prove that q is (k, c)-winning as follows: we use the policy with the properties from the claim, and switch to the policies that exist by the inductive hypothesis for the states that are $(k - 1, 0)$-winning or $(k, c' + 2)$-winning when the second condition from the claim is used. When another (k, c)-winning state is visited, we instead continue with a policy constructed from q' in the same way as for q. The fact that this composition of the policies yields a correct (k, c)-winning policy follows by induction: at every policy prefix t with $l'(t) = \mathbf{true}$ or $t = \epsilon$ such that no transition to a $(k - 1, 0)$-winning or $(k, c' + 2)$-winning has yet occurred, we know that the policy reaches some next goal state with probability at least p. For the other goal states, the correctness follows from the inductive hypothesis and the fact that after transitions to $(k - 1, 0)$-winning or $(k, c' + 2)$-winning goal states, the existing p-risk-averse policies can be used from there. If no $(k - 1, 0)$-winning or $(k, c' + 2)$-winning goal state is reached or until such a goal state is reached, the construction ensures that the goal states otherwise reached are (k, c)-winning, and no odd color higher than c is reached in between two visits to (k, c)-winning goal states that are not $(k - 1, 0)$-winning or $(k, c' + 2)$-winning. As this property holds (by induction over the length of the policy prefix) for all visits to goal states, the claim follows. ■

The characterization of (k, c)-winning states in Lemma 1 allows us to compute the (k, c)-winning states using traditional MDP policy computation algorithms.

Lemma 2: Let $\mathcal{M} = (S, A, \Sigma, P, C, s_0)$ be a parity MDP, $S_{k,c} \subseteq S$ be the (k, c)-winning states, $S_{k,c+2}, \ldots, S_{k,c_{maxEven}}$ be the $(k, c+2)$-winning to $(k, c_{maxEven})$-winnings states, and $S_{k-1,0}$ be the states that are $(k-1, 0)$-winning (all for some value of p). We can compute a reachability MDP $\mathcal{M}'$ with $|S| \times |\{c, c+2, \ldots, c_{maxEven}\}|$ many states in which the value of any state (q, c) is $\geq p$ if and only if q is a (k, c)-winning state.

Proof: We can construct $\mathcal{M}' = (S', A, \Sigma, P', g, s_0)$ as follows:

$$S' = S \times \{c, c+2, \ldots, c_{maxEven}\}$$

$$P((s,\tilde{c}),a)((s',\tilde{c}')) = \begin{cases} P(s,a)(s') & \text{if } C(s') \text{ is odd and} \\ & \tilde{c}' = \max(\tilde{c}, C(s')) \\ P(s,a)(s') & \text{if } C(s') \text{ is even and} \\ & \tilde{c}' = \tilde{c} \\ 0 & \text{else} \end{cases}$$
$$\text{for all}(s,\tilde{c}),(s',\tilde{c}') \in S', a \in A$$

$$g((s,\tilde{c})) = \begin{cases} 1 & \text{if } \tilde{c} = c, s \in S_{k,c}, C(s) \geq \tilde{c}, C(s) \text{ is even} \\ 1 & \text{if } s \in S_{k,\tilde{c}} \text{ or } s \in S_{k-1,0}, \text{ and } C(s) \text{ is even} \\ 0 & \text{else} \end{cases}$$
$$\text{for all}(s,\tilde{c}) \in S'$$

The MDP has the stated properties by the facts that (1) it keeps track of the highest color visited along a trace so far, and (2) it induces a payoff of 1 exactly for the states that are possible goal states. ■

Optimal policy computation for a reachability MDP can be performed by standard policy iteration or value iteration algorithms. Until now, the definition of the reachability MDP in Lemma 2 is somewhat recursive: in order to determine which states are (k, c)-winning, we have to already know the (k, c)-winning states. The characterization from Lemma 1 however allows us to compute it with the approach from Lemma 2. What we are actually searching for is the *largest* set of states $S_{k,c}$ that the construction from Lemma 2 maps to itself; any state set that is smaller misses some states that are (k, c)-winning by the characterization from Lemma 1, and by the same lemma, any set that is larger contains some state that is not (k, c)-winning. So computing the *greatest fixpoint* over the states $Q_{k,c}$ allows to find the (k, c)-winning states, provided that the $(k, c+2)$-winning to $(k, c_{maxEven})$-winning and $(k-1, 0)$-winning states are known. By iterating over the possible values of k and c, we can thus compute the sets $S_{k,c}$ in a bottom-up fashion, as shown in Algorithm 1.

The algorithm calls the external function COMPUTESTATEVALUES to solve the reachability MDPs obtained by the construction in Lemma 2, which can be a value or policy iteration algorithm. Extending Algorithm 1 to also compute a policy is simple: without loss of generality, optimal reachability MDP policies are positional, and we can stitch these

Algorithm 1 Algorithm to compute if a parity MDP $\mathcal{M}$ admits a p-risk-averse policy.

```
1:  function COMPUTERAPOLICY($\mathcal{M}, p$)
2:      $S_{k-1} \leftarrow \emptyset$
3:      while fixed point of $S_k$ has not been reached do
4:          for $c \in \{c_{maxEven}, c_{maxEven} - 2, \ldots, 0\}$ do
5:              $S_k[c] \leftarrow S$
6:              while fixed point of $S_k[c]$ has not been reached do
7:                  $\mathcal{M}' =$ CONSTRUCTIONFROMLEMMA2($c$, $S_k[c], \ldots, S_k[c_{maxEven}], S_{k-1}$)
8:                  $V \leftarrow$ COMPUTESTATEVALUES($\mathcal{M}'$)
9:                  $S_k[c] \leftarrow \{s \in S \mid V((s, c)) \geq p\}$
10:         $S_{k-1} \leftarrow S_k[0]$
11:     return $s_0 \in S_{k-1}$
```

policies together. The resulting policy always maintains a pair (k, c) and executes the positional policy computed by COMPUTESTATEVALUES in the respective iteration of the inner WHILE loop until a state in $S_k[c+2]$ or S_{k-1} is reached. In such cases, the policy then changes its pair to $(k, c+2)$ or $(k-1, 0)$ and continuous with the respective positional policy computed by COMPUTESTATEVALUES. Since the algorithm performs only a finite number of iterations over k and c, the resulting policy is finite-state.

Remark 1: To speed up Algorithm 1, we can simplify the reachability MDP construction of Lemma 2: instead of keeping track of the maximum odd color seen along a trace so far (in excess of c), we can alternatively keep track of whether an odd color greater than c has been seen so far, and only consider switching to a $(k-1, 0)$-winning goal state in that case. While the number of loop iterations of the algorithm until all positions that admit a p-risk-averse policy has been found can be higher with this modification, the reachability MDPs are typically smaller (as they have a size of at most $2 \cdot |S|$ then), which speeds up the value or policy iteration process for solving them.

B. Maximally risk-averse policy computation

In the previous subsection, we gave an algorithm to obtain p-risk-averse policies for a given p whenever they exist. In order to compute optimally risk-averse policies, we can apply a *bisection search*, which is the continuous-domain version of binary search, to find the highest value p such that a p-risk-averse policy exists.

Since p is a continuous value, this process has no natural termination point, however. For all practical means, it makes sense to define a cut-off value for the search such that if the difference between known upper and lower bounds on the risk-averseness level of the optimal policy is below the cut-off, the search process terminates with the best policy found until then. Defining a cut-off point is also motivated by practical means: most MDP solving algorithms run with a bounded precision, which leads to rounding errors. This makes it difficult to solve the problem given in Definition 3 in the strict sense.

However, under the assumption that the probabilities computed by function COMPUTESTATEVALUES are exact, Algorithm 1 can be modified in order to allow finding a maximally risk-averse policy. For this, line 9 of the algorithm needs to be replaced by $S_k[c] \leftarrow \{s \in S \mid V((s,c)) > p\}$. The algorithm then checks if a p'-risk-averse policy for $p' > p$ exists. Furthermore, after every call to COMPUTESTATEVALUES, we let the algorithm also compute $lb := \min\{V(s) \mid s \in S, V((s,c)) > p\}$. The least of these lb values represents a lower bound on the p-risk averseness of the policy actually computed. Let this value be named lb_{min}.

We can now perform an iterative search process for the optimally risk-averse policy as follows: starting with $p = 0$, we search for a p'-risk-averse policy for $p' > p$ using the modified version of Algorithm 1. If we find one, we update p to lb_{min} and continue with the search. Otherwise, the previously found policy is an optimally risk-averse policy.

To see why this process solves the problem, note that whenever p is increased, at least one state is removed from $S_k[c]$ in some iteration of the outermost while loop. While the state may be added to $S_k[c]$ *later* in the process, increasing the value of p can only push states to be found later in the search process of Algorithm 1. When delaying the addition of states to $S_k[c]$, at some point, there will be one execution of the outer while loop of Algorithm 1 in which no additional states are found. Since the algorithm will terminate without finding a policy in this case, by the correctness of the algorithm, we can terminate the search at that point, and the policy found last is optimally risk-averse.

VI. EXPERIMENTS

We implemented the p-risk-averse policy computation approach in a prototype tool, called `ramps`, written in `C++`. The tool uses the simplification from Remark 1 and employs value iteration to compute policies for the reachability MDPs analyzed in Algorithm 1. Bisection search with a cut-off value of 0.01 (i.e., 1 percent) is used to compute close-to-optimal risk-averse policies. We configured the value iteration processes to terminate when the sum of updates to the state values falls below 0.05. Value iteration is performed in a parallelized way using the `openmp` library. All computation times reported in the following were taken on an Intel i5-4200U computer with 1.60 GHz clock rate and 4GB of RAM, utilizing 2 physical processor cores, each with two virtual hyper-threaded cores used for value iteration. The `ramps` tool is available under the GPLv3 open-source license from `https://github.com/progirep/ramps`.

A. Single-robot control

In the first experiment, we consider the setting from Example 1. The `ramps` tool needs 30 minutes and 11 seconds (95m57s of single-processor time) to compute a 0.890689-risk-averse policy with 388329 states. A simulation of it, available as a video on `https://progirep.github.io/ramps`, shows that the robot performs the task encoded into the parity automaton until it crashes. Visiting the regions in the middle in the correct order seems to be relatively easy for the policy. In order to reach the regions on the left and on the right in a risk-averse way, the robot often circles many times before it has the right approach angle and position to travel through one of the gaps next to the static obstacles.

B. Multi-robot control

As a second example, we considered a multi-robot control scenario, which we depict in Figure 4. This time, we have two robots without complex dynamics: in each step, they can either move left, right, up or down by one cell, or choose not to move at all. If a robot chooses to move, there is an 8 percent chance that it moves into a different direction than chosen (i.e., $\frac{8}{3}$ percent per remaining direction). As in the first example, crashing into an obstacle or into the workspace boundaries leads to a transition to an error state in the MDP. A robot crashing into the other robot also leads to the error state.

The robots can also carry an item. For this, they have to jointly perform a pickup operation while standing left and right, respectively, of the pickup region r_1. While they maintain a horizontal distance of 2, they can continue carrying the item. The item is lost if there is a deviation in the distance. At region r_2, they can also drop the item. They cannot crash into each other while carrying an item (as it acts like a spacer). The MDP has 12294 states, 307304 state/action pairs, and 2798040 edges. The numbers of state/action pairs and edges are higher than in the first scenario, as each of the two robots has five choices of actions in each step.

The specification is represented as a 5-state parity automaton that encodes that (1) infinitely many items shall be delivered from r_1 to r_2, (2) infinitely often, robot one and two shall visit the top left and top right regions, respectively, and (3) the pickup and dropping regions should never be visited by any robot.

Computing a 0.599408-risk-averse policy takes 146.4 seconds and the simulation (available as a video on `https://progirep.github.io/ramps`) shows that again, the policy lets the robots perform their task until at least one of them collides. In case the item is lost during delivery, the two robots just try again immediately. The generated policy has 61509 states.

VII. CONCLUSION

In this paper, we showed how to compute *risk-averse* policies. A system governed by such a policy works towards the satisfaction of some given ω-regular specification even in probabilistic environments in which almost sure non-satisfaction of the specification cannot be avoided in the long run. Instead of just resigning, because the probability mass of the runs of a Markov decision process that satisfy the specification can only be 0, a p-risk averse policy always reaches the respective next *goal state* with a probability of at least p (from the previous goal state). The definition of the problem ensures that the goal states are chosen in a way that faithfully captures the satisfaction of the specification. We assumed that the specification is given as a deterministic

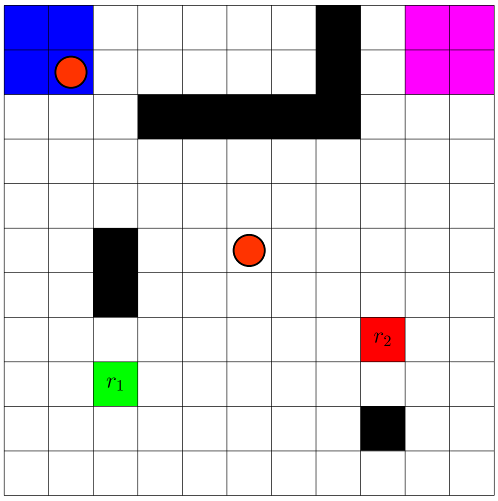

Fig. 4. Workspace for the multi-robot example.

parity automaton, but structured logics such as linear time logic (LTL) could also be used, as translations from LTL to parity automata are known [17].

We intent to extend the approach to the synthesis of strategies in stochastic two-player games in future work. Also, we will explore how to incorporate additional optimization criteria such as mean-average cost into policy generation and if reinforcement learning techniques can be used to successively approximate optimal policies during policy execution.

Acknowledgements

R. Ehlers was supported by the Institutional Strategy of the University of Bremen, funded by the German Excellence Initiative. S. Moarref and U. Topcu were partially supported by awards AFRL $FA8650$-15-C-2546, ONR $N000141310778$, ARO $W911NF$-15-1-0592, NSF 1550212, and DARPA $W911NF$-16-1-0001.

References

[1] X. Ding, S. L. Smith, C. Belta, and D. Rus, “Optimal control of Markov decision processes with linear temporal logic constraints,” *IEEE Trans. Aut. Control*, vol. 59, pp. 1244–1257, 2014.

[2] M. Lahijanian, S. B. Andersson, and C. Belta, “Temporal logic motion planning and control with probabilistic satisfaction guarantees,” *IEEE Trans. Robot.*, vol. 28, no. 2, pp. 396–409, 2012.

[3] S. Temizer, M. J. Kochenderfer, L. P. Kaelbling, T. Lozano-Pérez, and J. K. Kuchar, “Collision avoidance for unmanned aircraft using Markov decision processes,” in *AIAA Guidance, Navigation, and Control Conference*, 2010.

[4] R. Alterovitz, T. Siméon, and K. Y. Goldberg, “The stochastic motion roadmap: A sampling framework for planning with Markov motion uncertainty.” in *RSS*, vol. 3, 2007, pp. 233–241.

[5] B. Defourny, D. Ernst, and L. Wehenkel, “Risk-aware decision making and dynamic programming,” in *NIPS 2008 Workshop on Model Uncertainty and Risk in RL*, 2008.

[6] A. Condon, “The complexity of stochastic games,” *Information and Computation*, vol. 96, no. 2, pp. 203–224, 1992.

[7] L. De Alfaro, T. A. Henzinger, and O. Kupferman, “Concurrent reachability games,” in *FOCS*. IEEE, 1998, pp. 564–575.

[8] L. De Alfaro and T. A. Henzinger, “Concurrent omega-regular games,” in *15th Annual IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 2000, pp. 141–154.

[9] L. de Alfaro and R. Majumdar, “Quantitative solution of omega-regular games,” in *STOC*. ACM, 2001, pp. 675–683.

[10] K. Chatterjee, L. de Alfaro, and T. A. Henzinger, “The complexity of stochastic Rabin and Streett games,” in *ICALP*, 2005, pp. 878–890.

[11] K. Chatterjee, L. De Alfaro, and T. A. Henzinger, “The complexity of quantitative concurrent parity games,” in *17th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2006, pp. 678–687.

[12] K. Chatterjee and T. A. Henzinger, “A survey of stochastic ω-regular games,” *Journal of Computer and System Sciences*, vol. 78, no. 2, pp. 394–413, 2012.

[13] K. Chatterjee, M. Jurdziński, and T. A. Henzinger, “Quantitative stochastic parity games,” in *SODA*, 2004, pp. 121–130.

[14] X. C. Ding, S. L. Smith, C. Belta, and D. Rus, “LTL control in uncertain environments with probabilistic satisfaction guarantees,” in *18th IFAC World Congress*, 2011.

[15] M. Svorenova, I. Cerna, and C. Belta, “Optimal control of MDPs with temporal logic constraints,” in *52nd IEEE Conference on Decision and Control (CDC)*, 2013, pp. 3938–3943.

[16] O. Sigaud and O. Buffet, *Markov Decision Processes in Artificial Intelligence*. Wiley-IEEE Press, 2010.

[17] N. Piterman, “From nondeterministic Büchi and Streett automata to deterministic parity automata,” *Logical Methods in Computer Science*, vol. 3, no. 3, 2007.