Safe Sequential Decision Making in Uncertain Environments

A Dissertation

Presented to

the Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the requirements for the Degree

Doctor of Philosophy (Computer Science)

by

Ingy ElSayed-Aly

May 2024

Abstract

Sequential decision making is a collection of techniques automating decisions for a given system model. In general, exact models are unrealistic since real-world behavior may be influenced by external factors and modeling errors. We focus on methods which model uncertain environments featuring both stochasticity and non-determinism. This dissertation develops safe sequential decision making for different types of uncertain environments. We concentrate our efforts on two types of decision making: multi-agent reinforcement learning (MARL) where agents seek to learn optimal policies; and probabilistic model checking where the model's transitions and states are known. To address safe sequential decision making, we develop a suite of novel techniques to bring logic guided learning to MARL algorithms and informative distributional algorithms to probabilistic model checking.

In the first chapter, we introduce two novel shielding approaches for safe MARL synthesized based on a temporal logic safety specification. For our centralized approach, we synthesize a single shield to monitor all agents' joint actions and correct any unsafe actions. We also propose a factored shield where we synthesize multiple shields based on a factorization of the joint state space; the set of shields monitors agents concurrently. The factored approach has the advantage of being more scalable while the centralized shield is closer to optimal.

The shielding methods require general information about the underlying model which is not always available. Therefore, in the second chapter, we propose a novel framework for designing reward functions for agents to learn the desired behaviors while avoiding unsafe situations. We use temporal logic to define which behaviors to encourage or shun. We present a semi-centralized logic-based learning algorithm for reward shaping that is scalable in the number of agents.

Next, in the third chapter, we extend safe sequential decision making to probabilistic model checking. Specifically, we design a distributional extension to probabilistic model checking. We reason about a variety of distributional measures and propose a method to precisely compute the full distribution. We also approximate the optimal policy using distributional value iteration for varying levels of risk-sensitivity.

Finally, in the fourth chapter, we extend this work and formulate an algorithm to optimize the weighted expected value of accumulated rewards in uncertain parametric models. We leverage the joint distribution over the uncertain parameters using tractable yet expressive distributional representations to provide less conservative policies.

We evaluate the approaches implemented in this dissertation across a diverse range of relevant case studies. Experiments demonstrate significant advances in safe yet scalable methods for multi-agent reinforcement learning and informative safety policy analysis for probabilistic model checking. Moreover, to facilitate collaboration and future innovation, all frameworks developed are made publicly available.

Acknowledgements

First and foremost, I want to thank my advisor Lu Feng for her insight and invaluable guidance through thick and thin for all these years. I can't highlight enough how important her encouragement and kindness was to me when I felt like I couldn't take one step further. I'm honored to have been one of her PhD students.

My deepest gratitude to David Parker for being a mentor and collaborator for the second half of my PhD journey. For generously sharing his time and expertise. I would like to thank Matthew Dwyer, Nicola Bezzo and Shangtong Zhang for their valuable time and feedback as part of my PhD committee. Many thanks to the awesome collaborators I have had the honor to work with including Ufuk Topcu, Rüdiger Ehlers, Christopher Amato, Suda Bharadwaj. Thanks to all the developers who worked on PRISM which was instrumental to my research.

Thanks also to my friends from France, Egypt and beyond, Shehab Abd El Salam, Cinderella Amir, Gaston Ocampo, Sofia ElOthmani, Pauline Lobies, Pierre El Sayed Youssef and Alexis Walid Ahmed. Special thanks to Cécile, Ludovic and Éléonore Bernard for their friendship in Toulouse and for helping me cross the finish line. Thanks to all the people I met at UVA along the way who made my time there so much better especially Nabeel Nasir, Mahmoud Al Naggar, Aidan San, Phil Seaton, Aron Harder, Stuart Graham, Carl Hildebrandt and Yasunari Kato. Thanks to Arash Takavoli for being a beacon of optimism and generosity. Special thanks to Anna and Tyler Spears for refilling my social battery and for their unending compassion. Thanks to the Virginia Eventing and Dressage team which helped get back on my feet, Mimi Combs, JT Zarate, the owners and staff of Brookhill Farm, the lovely horses Banjo, Bailey, Ned and Tori. Special thanks to my labmates Meiyi Ma, Shili Sheng, MJ Kwon, Shuyang Dong and Kayla Boggess. I am most grateful for my dearest friends, roommates, travel buddies and partners in crime Josephine and Steven Lamp.

I want to give a huge thanks to the professors at the American University in Cairo without whom I would not be finishing a PhD program today including Mariam Ayad, Mohamed Shalan, Sherif El-Kassass, Mohamed Moustafa and Khaled el-Ayat. Special thanks to Ahmad Saqfalhait for his kindness and unending enthusiasm. Thanks all who believed in my research potential at EPFL including Serge Vaudenay, Mathilde Igier, Bogdan Ursu,

Damian Vízar and Sonia Bogos. My heartfelt thanks to Larry and Carol Walker for their endless generosity and kindness. I am also deeply grateful to Anil Shende for his unwavering support and his encouragement to pursue a PhD in the first place. Immense thanks to the Computer Science Department at the University of Virginia, especially Sandhya Dwarkadas, Hongning Wang, Haifeng Xu, Jack Davidson, Brad Campbell, Yuan Tian. I would also like to acknowledge the department's staff for going above and beyond especially Jai Maupin and Marion Hight. Special thanks to Kevin Skadron for being an awesome Department Chair during the trying times of the pandemic and for his support of new students. I would like to extend my sincere appreciation for the staff and doctors at the UVA hospital for being there every step of my recovery and especially Randy Ramcharitar, Deborah Grotenhuis, Louise Man, Luke Wilkins and Kathleen McManus. Special thanks to Madeline Dillon for her care, compassion and understanding.

I am indebted to Scarlet and Dave Courtois for many Christmas gatherings, books, bubble teas, and friendship above all. Thanks to Regina and Anthony for listening to my rambling and my doubts while sharing their own PhD experience with me. Thanks to BJ, Daly and Luke for welcoming me and sharing their home for many a thanksgiving. Thanks to Daniel for unexpected Richmond lunches and lively discussions. My sincere appreciation to Brian and Ludy for their support, kindness, courage and endless optimism. My eternal gratitude goes out to my partner Derek Lafever for standing by my side rain or shine. For putting a smile on my face every time, filling my life with love and adventure.

Last but not least, I want to thank my family. My heartfelt thanks to both my grandparents Jacqueline and Michel Bois for always having my back. I also want to thank my extended family for always being there for me especially my great aunts and uncles. To my late grandma who has always been a source of positivity and love. My brother for inspiring my determination and encouraging me to challenge myself. My sister, who keeps me grounded and whose courage has kept me going. Jazz, for his boundless companionship and support. Loza, for bringing joy and silliness to our family. Last but not least, I am forever indebted to my mom who has been both my staunchest supporter and my inspiration every step of the way.

To my family near and far,

Contents

List of Figures

List of Tables

List of Abbreviations

CPS	**C**yber **P**hysical **S**ystems
DFA	**D**eterministic **F**inite **A**utomaton
DTMC	**D**iscrete **T**ime **M**arkov **C**hain
LDBA	**L**imit-**D**eterministic **B**üchi **A**utomaton
LTL	**L**inear **T**emporal **L**ogic
MARL	**M**ulti **A**gent **R**einforcement **L**earning
MDP	**M**arkov **D**ecision **P**rocess
RL	**R**einforcement **L**earning
TL	**T**emporal **L**ogic
VI	**V**alue **I**teration

Chapter 1

Introduction

Cyber-physical systems (CPS) are becoming increasingly complex to better integrate with sensing, control and actuation of multiple devices in the physical world. Many of these devices provide safety-critical services including autonomous driving, data transfer, personal robots, and medical equipment. Sequential decision making is a collection of techniques automating decisions for a given model of a CPS and environment combination. In general, exact models are unrealistic since both system and environment behavior may be influenced by external factors and modeling errors. For example, two cars of the same make and model might still present slightly different behaviors or a drone performing surveillance might encounter unexpected winds. Therefore, we focus on methods which model uncertain environments featuring both stochasticity and non-determinism such as Markov Decision Processes (MDPs). We further concentrate our efforts on the following two classes of sequential decision making techniques: multi-agent reinforcement learning (MARL) where agents seek to learn optimal policies when the underlying model is largely unknown; and probabilistic model checking where the MDP's transitions and states are known enabling in-depth analysis. The main goal of this thesis is to help bridge the gaps between safe policy synthesis methods in MARL and probabilistic model checking.

Multi-agent reinforcement learning addresses sequential decision-making problems where multiple agents interact with each other in a common environment. In recent years, MARL methods have been increasingly used in a wide range of safety-critical applications from traffic management [1] to robotic control [2] to autonomous driving [3]. Existing MARL methods [4], [5] focus mostly on optimizing policies based on returns, none of which can guarantee safety (e.g., ensure no unsafe states are ever visited) during the learning process. Nevertheless, learning with provable safety guarantees is necessary for many safety-critical MARL applications where the agents (e.g., robots, autonomous cars) may break during the exploration process, leading to catastrophic outcomes. However, to accommodate the scale of these problems, recent MARL algorithms often trade safety for efficiency.

Probabilistic model checking offers a collection of techniques for modelling systems that exhibit probabilistic and non-deterministic behavior. It supports not only their verification against specifications in temporal logic, but also synthesis of optimal controllers (policies). Common models such as discrete-time Markov chains (DTMCs) and Markov decision processes have been used to model network protocols [6], [7], hardware properties, energy grid management and several robotic applications [8]. A range of verification techniques for these, and other models, are supported by widely used probabilistic model checkers such as PRISM [9] and Storm [10]. To capture the range of quantitative correctness specifications needed in practice, it is common to reason about rewards (or, conversely, costs) associated with a model. Examples include checking the worst-case execution time of a distributed coordination algorithm, or synthesizing a controller that guarantees the minimal energy consumption for a robot to complete a sequence of tasks. Typically the expected (average) value of these quantities is computed, however in some situations it is necessary to consider the full probability distribution. Notably, in safety-critical applications, it can be important to synthesize risk-sensitive policies, that avoid high-cost, low-probability events, which can still arise when minimizing policies for the expected value.

Research Questions: *How do we ensure safe sequential decision making in uncertain environments?* This topic is especially challenging since the definition of safety varies across different domains. We further define two sub-questions as follows.

Firstly, *how can we provide safety in a multi-agent environment where the underlying MDP is unknown?* If too little information is known about the system it becomes impossible to prevent worst case outcomes. A key challenge for this direction is to develop methods that learn to avoid clearly defined unsafe events with a limited amount of information or none at all while being scalable in the number of agents.

Secondly, *how can we extend more rigorous probabilistic model checking methods to accommodate for more notions of safety and risk?* One aspect of the difficulty here is to propose general frameworks that support multiple types of queries and further analysis of the results. Furthermore, adapting these frameworks to handle more uncertain models is also important.

1.1 Dissertation Overview

While significant research progress has targeted the scalability of reinforcement learning and probabilistic model checking tools, only a few aspects of safety have been explored. Moreover, safety work for MARL and model checking have evolved independently without considering the advances made within the context of the other. MARL has mostly

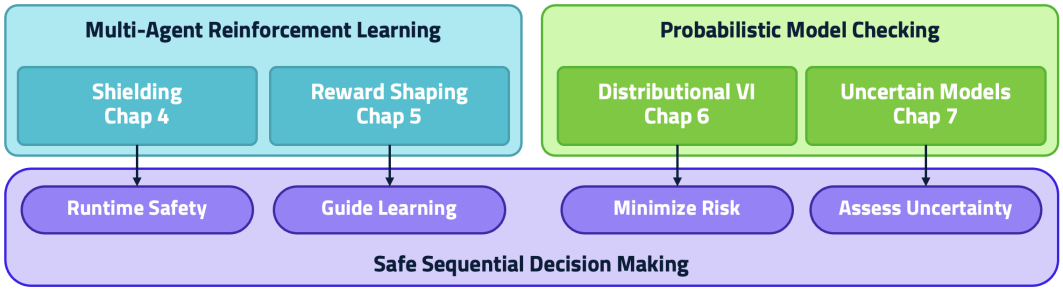


FIGURE 1.1: Dissertation Overview

focused on steering the policies with ad-hoc reward functions, ignoring the provably safe techniques from synthesis and model checking tools. Recent endeavors in probabilistic model checking involve scalability of the existing tools and risk sensitivity in the form of specialized methods that only target a single metric at a time.

Dissertation Statement. In this dissertation, we develop novel safe sequential decision making methods based on multi-agent reinforcement learning and probabilistic model checking. For multi-agent reinforcement learning, the main objective is to mitigate the risk for unsafe behaviors using tools from formal methods. For probabilistic model checking, the aim is to augment the algorithms with additional information to assess the risk and uncertainty levels of the resulting solutions.

Figure 1.1 highlights the general organization of the methods presented in this thesis. Each chapter's methods handles varying levels of uncertainty in the input model and targets different high-level safety objectives as shown in the corresponding ellipse. In what follows, we give a brief overview of each chapter presented in this dissertation.

1.1.1 Safe MARL via Shielding

In Chapter 4, we introduce two novel shielding approaches for safe MARL where each is synthesized based on a Linear Temporal Logic (LTL) safety specification. In both types of shields, the safety specification represents the set of unsafe behaviors to prevent. We then solve a safety game based on the provided specification and an abstraction of the MDP representing the agents' interactions with the environment. For our centralized approach, we synthesize a single shield to monitor all agents' joint actions and correct any unsafe action only if necessary. However, the centralized nature of this method means that its scalability in the number of agents is limited. For this reason, we further propose a factored shield where we synthesize multiple shields based on a factorization of the joint state space observed by all agents; the set of shields monitors agents concurrently and each shield is only responsible for a subset of agents at each step. The shields synthesized for each type allow us to ensure runtime safety by preventing agents' most

unsafe behaviors during learning and execution while leveraging the scalable aspects of MARL. Experimental results show that both approaches can guarantee the safety of agents during learning without compromising the quality of learned policies.

1.1.2 Logic-guided MARL Reward Shaping

The shielding approaches described above provide strong safety guarantees, but the MDP abstraction used to synthesize the shields is not always available. For this chapter, we intend to reliably guide the MARL agents learned policies in environments where the underlying MDP is entirely unknown. In fact, MARL agents rely heavily on exploration to learn from their environment and maximize observed rewards. It is, therefore, essential to design reward functions which lead each agent to learn the desired behaviors defined as an LTL specification. In this case, the specification represents the desired behavior rather than the unsafe behavior as is done in Chapter 4. Previous work has combined automata and logic based reward shaping with environment assumptions to provide an automatic mechanism to synthesize the reward function based on the task specification. However, there is limited work on how to expand logic-based reward shaping to the multi-agent setting. Chapter 5 presents a novel method for semi-centralized logic-based MARL reward shaping that is scalable in the number of agents. Experiments in multiple environments and multiple tasks show improved learning performance using our method.

1.1.3 Distributional Probabilistic Model Checking

While multi-agent reinforcement learning proposes algorithms supporting large unknown environments, it is not currently very compatible with rigorous analysis of the output policy. Verifying and providing formal guarantees for reinforcement learning is currently ongoing research. On the contrary, probabilistic model checking allows for in-depth analysis of MDPs through a wide range of quantitative properties. However, most methods usually work with respect to the expected value of these quantities, which can mask important aspects of the full probability distribution such as the risk of high cost events. In Chapter 6, we present a distributional extension of probabilistic model checking, for discrete-time Markov chains and Markov decision processes. We formulate distributional queries, which can reason about a variety of distributional measures, such as variance, value-at-risk or conditional value-at-risk, for the accumulation of reward or cost until a co-safe linear temporal logic formula is satisfied. For DTMCs, we propose a method to compute the full distribution to an arbitrary level of precision. For MDPs, we approximate the optimal policy using distributional value iteration. We implement our techniques and investigate their performance and scalability across a range of large benchmark models.

1.1.4 Distributionally Uncertain Probabilistic Model Checking

Distributional probabilistic model checking allows for a more general framework for assessing a variety of distributional metrics such as risk-specific values. However, requiring full information about the transitions of the MDP models to be evaluated is impractical in some cases. Previous work on parametric MDPs shows how uncertainty in the parametric transition probabilities of the model can lead to a large set of possible solutions. Moreover, few methods provide ways to tractably represent and assess the effects of this uncertainty. In Chapter 7, we introduce an extension to the work in Chapter 6 in the form of a novel algorithm for uncertain parametric MDP models featuring a joint distribution over the parameters. A key ingredient to this approach is the use of tractable yet expressive distributional representations for parameter uncertainty. Our approach optimizes the weighted expected value of the accumulated rewards over these models resulting in a policy exploiting the probabilities of the parameter realizations. Experimental results show that our approach can provide additional information about the resulting policies while being less conservative than comparable baseline methods.

1.2 Summary of Contributions

In this dissertation, we develop several algorithms for safe sequential decision making using multi-agent reinforcement learning and probabilistic model checking. The main contributions are summarized as follows:

- Bringing state of the art safe policy synthesis methods to MARL and probabilistic model checking.
- Developing centralized and factored shielding approaches to prevent unsafe outcomes in MARL.
- Presenting a scalable semi-centralized logic-based reward shaping method.
- Building a general formal framework for distributional probabilistic model checking to allow for the assessment of distributional metrics.
- Designing a risk-sensitive distributional value iteration algorithm compatible with the previously mentioned framework.
- Devising an algorithm for distributionally uncertain parametric MDPs to assess the weighted possible outcomes based on the parameter uncertainty.

1.3 Dissertation Structure

An illustration of this dissertation's structure is provided in Figure 1.1. Chapter 2 reviews existing research related to that presenting in this dissertation. In Chapter 3, I introduce the necessary background for the following chapters. My main contributions are presented in Chapters 4-7. Finally, I summarize my research, discuss future directions and broader impacts in Chapter 8.

1.4 Other Publications and Credits

Parts of this work was jointly developed with other researchers and published as such. However, I am the main contributor for the research presented in this dissertation and credit my collaborators in the following.

Chapter 4 is based on the work in [11] published along with Suda Bharadwaj, Christopher Amato, Rüdiger Ehlers, Ufuk Topcu and Lu Feng at AAMAS'21. I was solely responsible for the development of this work while provided with starter synthesis code from Suda Bharadwaj and instructions on the use of the synthesizer from Rüdiger Ehlers. Christopher Amato, Rüdiger Ehlers, Ufuk Topcu and Lu Feng were instrumental in guiding my design of the algorithms proposed.

Chapter 5 presents the contents of the technical report [12] under the advisement of Lu Feng. Haiying Shen, Haifeng Xu and Hongning Wang have also provided feedback for this work.

Chapter 6 is based on joint work with David Parker and Lu Feng to be published in [13] . I actively worked with David Parker and Lu Feng to determine the distributional model checking framework and define the distributional queries. The implementation is based on the PRISM open-source model checking tool managed by David Parker. David Parker also developed the algorithm in Section 4. However, I was responsible for implementation the algorithms in Section 5 and Section 6 as well the evaluation and results.

Chapter 7 is based on joint research with David Parker and Lu Feng which is not yet published. The implementation is also based on the PRISM open-source model checking tool managed by David Parker. The code used as a baseline for comparison is part of the PRISM codebase as well. I was responsible for the design and implementation of the algorithm presented in Section 7.3.

Chapter 2

Related Work

In this chapter, we review previous works relevant to the subsequent chapters including works pertaining to safe reinforcement learning and shielding in Section 2.1, works relevant to reward shaping and multi-agent reinforcement learning in Section 2.2, works relating to distributional reinforcement learning and probabilistic model checking in Section 2.3 and finally works pertaining to robust planning and uncertain models in Section 2.4.

2.1 Safe Multi-Agent Reinforcement Learning via Shielding

Safe reinforcement learning (RL) is an active research area, but existing results focus mostly on the single-agent setting [14], while safe MARL is still relatively uncharted territory [5]. The survey in [14] classifies safe RL methods into two categories: (1) transforming the optimization criterion with a safety factor, such as the worst case criterion, risk-sensitive criterion, or constrained criterion; and (2) modifying the exploration process through the incorporation of external knowledge (e.g., demonstrations, teacher advice) or the guidance of a risk metric. Our shielding approaches fall into the second category. In particular, shields act similarly to a teacher who provides information (e.g., safe actions) to the learner when necessary (e.g., unsafe situations are detected). The concept of shielding was introduced to RL for the single-agent setting in [15]. In this work, we adapt the shielding framework for MARL by addressing challenges such as the coupling of agents and scalabilty issues in the multi-agent setting.

Different safety objectives for RL have been considered in the literature, such as the variance of the return, or limited visits of error states [14]. In this work, we synthesize shields that enforce safety specifications expressed in linear temporal logic (LTL) [16], which is a commonly used specification language in formal methods for safety-critical systems [17], [18]. For example, LTL has been used to express complex task specifications for robotic planning and control [19], [20]. Several recent works [21]–[23] have developed reward shaping techniques that translate logical constraints expressed in LTL to reward functions for RL. However, as we demonstrated in our experiments (Section 4.3), relying on

reward functions only is not sufficient for MARL methods to learn policies that guarantee the safety (e.g., no collisions).

The shield synthesis technique based on solving two-player safety games was developed in [24] for enforcing safety properties of a system at runtime, and was adopted in [15] to synthesize shields for single-agent RL. We further adapt this technique to synthesize centralized and factored shields for MARL in this paper. There are a few recent works [25], [26] considering the shield synthesis for multi-agent (offline) planning and coordination, none of which are directly applicable for MARL.

2.2 Logic-Guided Reward Shaping

Reward Shaping. Linear Temporal Logic (LTL) is a commonly used specification language in formal methods for safety-critical systems [17], [18]. For example, LTL has been used to express complex task specifications for robotic planning and control [19], [20]. Tasks expressed in LTL can be represented as a Limit-Deterministic Büchi Automaton (LDBA) clearly highlighting accepting states [21]. The product of the obtained LDBA and the environment MDP can be used to designed a logic based reward function. In Hasanbeig et al. [21] the reward function is purely based on visiting the accepting states whereas in Bozkurt et al. [22] the reward function is based on the path that is visited. In Kuo et al. [27] and in Elbarbari et al. [28], the authors explore different ways to capture metrics of progress within the logic synthesized automaton in order to help guide the learning more effectively.

Logic based reward shaping can also be used in continuous control use cases [29], [30]. In the context of RL, logic-based reward shaping has often been augmented by using control barrier functions to address probabilistic safety guarantees [21], [29], [30]. In the context of multi-agent systems, there are ways to plan using LTL without learning [19], [31]. There is one recent work considering safety in MARL by modifying the rewards the agents receives given the output of a logic-synthesized shield [11]. However, this work cannot be considered reward shaping because it does not guide the agent towards desired accepting states.

Some works consider the co-safe fragment of LTL and instead convert the tasks to Mealy Machines to construct so-called Reward Machines (RM) [32]. The work on RMs has recently been expanded to include cooperative MARL through task decomposition to individual agent tasks and decentralized Q-learning [33]. Because of the use of the RMs, the method is limited to the co-safe fragment of LTL in addition to considering only tasks that can be fully decentralized. This method also needs to decompose the LTL tasks in order for each agent to learn one specific task.

Multi-agent Reinforcement Learning. In Reinforcement Learning an agent learns an optimal behavior based on multiple trials. The agent interacts with an unknown environment usually modeled as an MDP. At each step, the agent chooses some action a, the environment moves it to the next state based on a transition probability and the agent receives a reward r. The aim of the agent is to adapt its behavior in order to maximize the expected return $\mathcal{R} = \Sigma_{t=0}^{\inf} \gamma^t r_t$ where $0 < \gamma \leq 1$ is a discount factor [34].

In multi-agent reinforcement learning (MARL), at each time step t, each agent selects an action simultaneously, this set of all agent actions is called the joint action. The environment will then return a reward for each agent. In MARL, the reward function may favor competition or cooperation [35] MARL algorithms can be considered Joint Action learners or Independent Learners [34], [35]. For Joint Action learning algorithms, each agent considers all other agents, leading to complete communication but also poor scalability. In practice, it is unlikely that agents will need to communicate and share information with all other agents at each step. Independent Learners consider other agents when they "detect" a need to coordinate or simply consider other agents part of the environment such in Independent Q-learning [36]. Interestingly, despite Q-functions learned in Independent Q-Learning relying only on one agent's awareness, Independent Q-Learning has successfully been applied in multiple multi-agent settings with some limitations [35], [37]. In the case of a larger state space, deep learning has proven to be a useful tool to generalize across states and reduces the need for manual feature design [4], [38].

2.3 Distributional Probabilistic Model Checking

Distributional properties. Some existing probabilistic model checking methods consider distributional properties beyond expected values, notably *quantiles* [39]–[42], i.e., optimal reward thresholds which guarantee that the maximal or minimal probability of a reward-bounded reachability formula meets a certain threshold. While [39] and [40] focus on complexity results, [41] and [42] consider practical implementations to compute quantiles, for single- and multi-objective variants, respectively, using model unfoldings over "cost epochs"; [42] also proposes the use of interval iteration to provide error bounds. By contrast, our methods derive the full distribution, rather than targeting quantiles specifically, and our DTMC approach derives error bounds from a forward computation. We also mention [43], which computes probability distributions in a forwards manner, but for infinite-state probabilistic programs and using generating functions, and [44], which proposes an algorithm (but not implementation) to compute policies that trade off expected mean payoff and variance.

Risk-aware objectives. For MDPs, we focus in particular on *conditional value-at-risk* (CVaR). There are alternatives, such as mean-variance [45] and value-at-risk [46] but, as

discussed in [47], these are not *coherent risk metrics*, which may make them unsuitable for rational decision-making. Other work on the CVaR objective includes: [48], which studies decision problem complexity, but for mean-payoff rewards and without implementations; [49], which repeatedly solves piecewise-linear maximization problems, but has limited scalability, taking over 2 hours to solve an MDP with about 3,000 states; and [50], which proposes both linear programming and value iteration methods to solve CVaR for MDPs and DTMCs. Other, not directly applicable, approaches tackle constrained problems that incorporate the CVaR objective [51]–[53]. Our approach differs from all these in that it computes the full distribution, allowing multiple distributional properties to be considered. We also work with temporal logic specifications. Alternative temporal logic based approaches to risk-aware control include [54], which proposes risk-aware verification of MDPs using cumulative prospect theory, and [55] which proposes chance constrained temporal logic for control of deterministic dynamical systems.

Distributional reinforcement learning. Our work is based on probabilistic model checking, which fully explores known models, but our use of DVI is inspired by *distributional reinforcement learning* [56], which can be used to learn risk-sensitive policies and improve sample efficiency (see [57] for a comparison of expected and distributional methods). We take a formal verification approach and use numerical solution, not learning, but adopt existing categorical and quantile distributional approximations and our risk-neutral DVI algorithm is a minimization variant adapted from [56]. Risk-sensitive DVI is also sketched in [56], based on [58], but only a theoretical analysis of the method is given, without considering practical implementation aspects, such as how to discretize slack variables for computational efficiency, and how such approximations would affect the correctness of model checking. We extend risk-sensitive DVI with a discretized slack variable and show its effects theoretically in Section 6.2.3 and empirically via computational experiments in Section 6.3.

2.4 Uncertain Probabilistic Model Checking

Robust Planning. Handling uncertainty in sequential decision making has traditionally involved statistical tools and probability theory. However, uncertainty stems from different sources such as errors in system modeling (systemic or epistemic uncertainty) and the unpredictability of real environmental factors (aleatoric uncertainty) [59], [60]. It stands to reason that explicitly modeling the two types of uncertainty can help us attempt to reduce the epistemic uncertainty while considering the irreducible aleatoric uncertainty. Recent work in Robust RL represents the epistemic uncertainty as a trained adversary to learn robust policies that perform better when transferred from simulation to real environments [61]. Additional work uses an MDP augmented with a belief distribution modeling

the epistemic uncertainty formulation called Bayes-Adaptive MDP to better trade-off exploration and exploitation [62]. A common objective of solutions considering uncertainty is to learn or synthesize *distributional robust* policies that achieve the maximum expected reward under the most adversarial realization of uncertain parameters. The notion of targeting worst-case possibilities is further developed in [63] where a risk metric is used to generate policies that are risk-averse to both aleatoric and epistemic uncertainty. In [64], the authors use apply a risk-averse metric to model uncertainty in combination with constrained RL algorithms to propose safer distributionally robust solutions.

Distributionally Robust MDP. One formulation often used for robust MDP planning is the Distributionally Robust MDP (DR-MDP) [65] which models the uncertainty using sets of possible distributions corresponding to different confidence levels inspired by the work on Robust MDPs [66]. [65] proposes a value iteration method with a minimax form that requires finding the worst-case distributions over the transition models. Subsequent work in that direction includes representing the uncertainty over the transition models (ambiguity set) using different techniques. [67] uses moment-based ambiguity sets that stores information related to moments of the distribution over the transitions. Distance-based ambiguity sets instead use the distributional distance such as the Wasserstein distance [68] or the Kullback-Leibler divergence [69] between a reference distribution and that observed from simulation or samples. There also exists work using confidence regions for the ambiguity sets [70]. Recent work presented in [71] studies the different types and provides hybrid methods using ambiguity sets containing both moment-based and distance-based information. Unfortunately, most methods using DR-MDP formulations tend to be overly conservative. [72] uses Bayesian posterior distributions for the ambiguity sets in the DR-MDP. They also allow for balancing the posterior expected performance and the robustness to adversarial realization by minimizing a risk metric on the possible adversarial distributions. The research around DR-MDPs is related to this work in the sense that we also explicitly represent the the distribution over the possible transitions as explained in section 7.2.

Uncertain Markov models. Another related line of work in model checking with uncertainty involves using uncertain Markov models (uMDP, uDTMC) some of which represent uncertainty over parameters which can control the transition models (pMDP, pDTMC). [73] gives an overview of the different techniques used when the objective is to find out which regions of the parameter space satisfy a temporal logic property if any. In [74], [75] scenario-based optimization on a pMDP is used to compute the probability that there exists a policy satisfying the specification for a given confidence value. In this case, the assumption is that the parameters are somewhat controllable so that removing certain regions can lead to a desirable outcome. In [76], the authors tackle the problem of finding the best method to sample from the parameter space to

obtain enough information about a markov chain. A parametric Markov chain can also be obtained by using Bayesian Networks where [77] uses model checking to find the parameter regions that satisfy a probabilistic LTL formula. [78] uses Bayesian Learning along with to return a satisfaction function over the parameter space based on a (limited) set of observations for a continuous-time markov chain and a temporal logic property. Bayesian Learning is also used in [79] to iteratively improve the information gathered about the parametric uMDP eventually converging to a standard MDP. Specifically, they use an uncertain MDP where the parameter uncertainty is expressed using intervals. Robust VI [80] is used to synthesize a policy that maximizes (or minimizes) the worst case performance for a given metric (e.g., expected reward). RL policies have also been converted to a DTMC abstraction [81] or to an interval MDP in [82] to determine the probability of reaching unsafe states. Robust VI is also used in [83] where a discrete abstraction in the form of an interval MDP models a continuous environment to synthesize a controller taking into account aleatoric and epistemic uncertainty.

To the best of our knowledge, previous work for robust probabilistic model checking has taken the approach of either finding the regions in the parameter space that satisfy a property or considered the uncertainty of the parameter to be controllable. This work leverages the previous model checking advances in Distributional VI [13] which uses tractable distributional representations of the aleatoric uncertainty to return a distribution over the costs. Our work extends the distributional VI ideas to the more general uncertain MDP context while more explicitly modeling the epistemic uncertainty also using discrete approximations for the parameter distributions. While our work focuses on synthesizing a policy for given parameter data, this work can be extended to be included within a learning loop where the uncertainty is refined.

Chapter 3

Preliminaries

In this chapter, we go over the necessary background for the subsequent chapters relating to multi-agent reinforcement learning and probabilistic model checking. We let $\mathbb{N}$, $\mathbb{R}$, and $\mathbb{Q}$ denote the sets of naturals, reals and rationals, respectively, and write $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$.

3.1 Multi-Agent Reinforcement Learning (MARL)

We define a discrete probability *distribution* over a (countable) set S as a function $\mu : S \to [0, 1]$ such that $\sum_{s \in S} \mu(s) = 1$. Let $Distr(S)$ denote the set of distributions over S.

We follow the Markov game formulation of MARL in [5]. A *Markov game* is a tuple $(N, S, \{A^i\}_{i \in N}, P, \{R^i\}_{i \in N}, \gamma)$ with a finite set $N = \{1, \cdots, n\}$ of agents, and a finite state space S observed by all agents; let $A := A^1 \times \cdots \times A^n$ be the set of joint actions for all agents, where A^i denotes the actions of agent $i \in N$; the probabilistic transition function $P : S \times A \to Distr(S)$ is defined over the joint states and actions of all agents; $R^i : S \times A \times S \to \mathbb{R}$ is an immediate reward function for agent i under the joint states and actions; $\gamma \in [0, 1]$ is the discount factor of future rewards. At time step t, each agent chooses an action $a_t^i \in A^i$ based on the observed state $s_t \in S$. The environment moves to state s_{t+1} with the probability $P(s_t, a_t, s_{t+1})$, where $a_t = (a_t^1, \cdots, a_t^n)$ is the joint action of all agents, and rewards agent i with $R^i(s_t, a_t, s_{t+1})$. The goal of an individual agent i is to learn a policy $\pi^i : S \to Distr(A^i)$ that optimizes the expectation of cumulative future rewards $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R^i(s_t, a_t, s_{t+1})]$. The performance of individual agent i is not only influenced by its own policy, but also the choices of all other agents.

Depending on agents' goals, MARL algorithms can be categorized as fully cooperative (i.e., agents collaborate to optimize a common long-term return), fully competitive (i.e., zero-sum game among agents), or a mixed setting that involves both cooperative and competitive agents. In our experiments (Section 4.3), we used the following three mixed-setting algorithms. Independent Q-learning [84] is a baseline algorithm where agents learn Q-values over their own action set independently and do not use any information about

other agents. CQ-learning [85] is an algorithm that allows agents to act independently most of the time and only accounts for the other agents when necessary (e.g., when conflict situations are detected). MADDPG [38] is a deep MARL algorithm featuring centralized training with decentralized execution, in which each agent trains models simulating each of the other agents' policies based on its observation of their actions.

Scalability is a key challenge of MARL, due to its combinatorial nature. For example, our experiments can only use two agents with CQ-learning, but more than four agents with MADDPG which applies deep neural networks for function approximation to mitigate the scalability issue. Another key challenge of MARL is the lack of convergence guarantees in general, except for some special settings [5]. As multiple agents learn and act concurrently, the environment faced by an individual agent becomes non-stationary, which invalidates the stationary assumption used for proving convergence in single-agent RL algorithms.

3.2 Probabilistic Model Checking

We begin with some background on random variables, probability distributions, and the probabilistic models used in chapters 6 and 7.

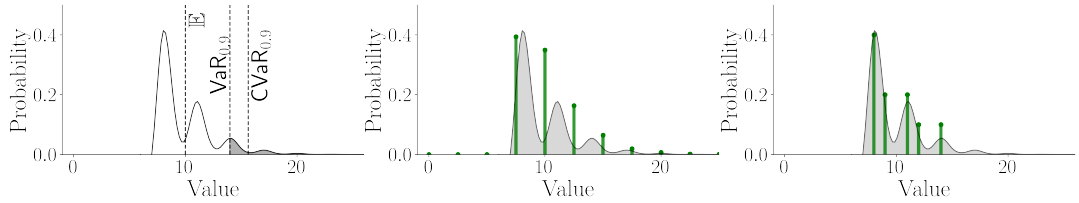
3.2.1 Random Variables and Probability Distributions

Let $X : \Omega \to \mathbb{R}$ be a random variable over a probability space $(\Omega, \mathcal{F}, \Pr)$. The *cumulative distribution function* (CDF) of X is denoted by $\mathcal{F}_X(x) := \Pr(X \leq x)$, and the inverse CDF is $\mathcal{F}_X^{-1}(\tau) := \inf\{x \in \mathbb{R} : \mathcal{F}_X(x) \geq \tau\}$. Common properties of interest for X include, e.g., the *expected value* $\mathbb{E}(X)$, the *variance* $\mathrm{Var}(X)$ which is the square of the *standard deviation* (s.d.), or the *mode*.

We also consider several *risk*-related measures. The *value-at-risk* of X at level $\alpha \in (0, 1)$ is defined by $\mathsf{VaR}_\alpha(X) := \mathcal{F}_X^{-1}(\alpha)$, which measures risk as the minimum value encountered in the tail of the distribution with respect to a risk level α. The *conditional value-at-risk* of X at level $\alpha \in (0, 1)$ is given by $\mathsf{CVaR}_\alpha(X) := \frac{1}{1-\alpha} \int_\alpha^1 \mathsf{VaR}_\nu(X) d\nu$, representing the expected loss given that the loss is greater or equal to VaR_α.

Example 1. Figure 3.1a illustrates an example probability distribution of a random variable X, annotated with its expected value $\mathbb{E}(X)$, value-at-risk $\mathsf{VaR}_{0.9}(X)$ and conditional value-at-risk $\mathsf{CVaR}_{0.9}(X)$. ■

When working with the probability distributions for random variables, we write distributional equations as $X_1 \overset{D}{:=} X_2$, denoting equality of probability laws (i.e., the random variable X_1 is distributed according to the same law as X_2). We use δ_θ to denote the Dirac delta distribution that assigns probability 1 to outcome $\theta \in \mathbb{R}$. In practice, even



(A) True distribution (B) Categorical ($m = 11$) (C) Quantile ($m = 10$)

FIGURE 3.1: An example distribution with its categorical and quantile representations.

when distributions are discrete, we require approximate, finite representations for them. In chapters 6 and 7, we consider *categorical* and *quantile* distributional representations, both of which provide desirable characteristics such as tractability and expressiveness [56].

Definition 1 (Categorical representation)**.** A *categorical representation* parameterizes the probability of m *atoms* as a collection of evenly-spaced locations $\theta_1 < \cdots < \theta_m \in \mathbb{R}$. Its distributions are of the form $\sum_{i=1}^{m} p_i \delta_{\theta_i}$ where $p_i \geq 0$ and $\sum_{i=1}^{m} p_i = 1$. We define the *stride* between successive atoms as $\varsigma_m = \frac{\theta_m - \theta_1}{m-1}$.

Definition 2 (Quantile representation)**.** A *quantile representation* parameterizes the location of m equally-weighted atoms. Its distributions are of the form $\frac{1}{m}\sum_{i=1}^{m} \delta_{\theta_i}$ for $\theta_i \in \mathbb{R}$. Multiple atoms may share the same value.

Next, we introduce two metrics for measuring the distance between probability distributions. Let $p \in [1, \infty)$ and μ, μ' be two distributions. The *p-Wasserstein distance* $w_p(\mu, \mu')$ and the *Cramér distance* $\ell_2(\mu, \mu')$ are defined as:

$$w_p(\mu, \mu') := \left(\int_0^1 |\mathcal{F}_\mu^{-1}(\tau) - \mathcal{F}_{\mu'}^{-1}(\tau)|^p d\tau \right)^{\frac{1}{p}}$$
$$\ell_2(\mu, \mu') := \left(\int_{\mathbb{R}} |\mathcal{F}_\mu(x) - \mathcal{F}_{\mu'}(x)|^2 dx \right)^{\frac{1}{2}}$$

Given two multi-dimensional distributions $\eta, \eta' \in \mathit{Dist}(\mathbb{R})^S$, we define the *supremum p-Wasserstein distance* between them as $\overline{w}_p(\eta, \eta') := \sup_{s \in S} w_p(\eta(s), \eta'(s))$ and the *supremum Cramér distance* as $\overline{\ell}_2(\eta, \eta') := \sup_{s \in S} \ell_2(\eta(s), \eta'(s))$.

We can then describe how to *project* a distribution onto a representation.

Proposition 1 (Categorical projection [56])**.** For a probability distribution μ, there exists a (unique) projection of μ in the Cramér distance (ℓ_2) onto the categorical representation denoted by $\Pi_{\text{C}}\mu = \sum_{i=1}^{m} p_i \delta_{\theta_i}$ with parameters $p_i = \mathbb{E}_{X \sim \mu}\big(h_i(\frac{X - \theta_i}{\varsigma_m})\big)$, where $h_i(x)$ are (half-)triangular kernel functions defined as:

$$h_1(x) = \begin{cases} 1 & x \leq 0 \\ \max(0, 1 - |x|) & x > 0 \end{cases} \quad h_m(x) = \begin{cases} \max(0, 1 - |x|) & x \leq 0 \\ 1 & x > 0 \end{cases}$$

and $h_i(x) = \max(0, 1 - |x|)$ for $i = 2, \ldots, m - 1$.

Proposition 2 (Quantile projection [56])**.** For a probability distribution μ, a projection of μ in the 1-Wasserstein distance (w_1) onto the quantile representation is given by $\Pi_Q \mu = \frac{1}{m} \sum_{i=1}^{m} \delta_{\theta_i}$ with parameters $\theta_i = \mathcal{F}_\mu^{-1}(\frac{2i-1}{2m})$.

Example 2. Figures 3.1b and 3.1c show categorical and quantile representations, respectively, approximating the distribution shown in Figure 3.1a. ■

3.2.2 Markov Chains and Markov Decision Processes

In this dissertation, we work with both discrete-time Markov chains (DTMCs) and Markov decision processes (MDPs).

Definition 3 (DTMC)**.** A *discrete-time Markov chain* is a tuple $\mathcal{D} = (S, s_0, P, AP, L)$, where S is a set of states, $s_0 \in S$ is an initial state, $P : S \times S \to [0, 1]$ is a probabilistic transition matrix satisfying $\forall s \in S : \sum_{s' \in S} P(s, s') = 1$, AP is a set of atomic propositions and $L : S \to 2^{AP}$ is a labelling function.

A DTMC $\mathcal{D}$ evolves between states, starting in s_0, and the probability of taking a transition from s to s' is $P(s, s')$. An (infinite) *path* through $\mathcal{D}$ is a sequence of states $s_0 s_1 s_2 \ldots$ such that $s_i \in S$ and $P(s_i, s_{i+1}) > 0$ for all $i \geq 0$, and a finite path is a prefix of an infinite path. The sets of all infinite and finite paths in $\mathcal{D}$ are denoted $IPaths_\mathcal{D}$ and $FPaths_\mathcal{D}$, respectively. In standard fashion [86], we define a probability measure $\Pr_\mathcal{D}$ over the set of paths $IPaths_\mathcal{D}$.

Definition 4 (MDP)**.** A *Markov decision process* is a tuple $\mathcal{M} = (S, s_0, A, P, AP, L,)$, where states S, initial state s_0, atomic propositions AP and labelling L are as for a DTMC, and $P : S \times A \times S \to [0, 1]$ is a probabilistic transition function satisfying $\forall s \in S, \forall a \in A : \sum_{s' \in S} P(s, a, s') \in \{0, 1\}$.

In each state s of an MDP $\mathcal{M}$, there are one or more *available* actions which can be taken, denoted $A(s) = \{a \in A \mid P(s, a, s') > 0 \text{ for some } s'\}$. If action a is taken in s, the probability of taking a transition from s to s' is $P(s, a, s')$, also denoted $P(s'|s, a)$. Paths are defined in similar fashion to DTMCs but are now alternating sequences of states and actions $s_0 a_0 s_1 a_1 s_2 \ldots$ where $a_i \in A(s_i)$ and $P(s_i, a_i, s_{i+1}) > 0$ for all $i \geq 0$, and the sets of all infinite and finite paths are $IPaths_\mathcal{M}$ and $FPaths_\mathcal{M}$, respectively.

The choice of actions in each state is resolved by a *policy* (or *strategy*), based on the execution of the MDP so far. Formally, a policy takes the form $\pi : \mathit{FPaths} \to A$. We say that π is *memoryless* if the mapping $\pi(\omega)$ depends only on $\mathit{last}(\omega)$, the final state of ω, and *finite-memory* if it depends only on $\mathit{last}(\omega)$ and the current memory value, selected from a finite set and updated at each step of execution. The set of all policies for MDP $\mathcal{M}$ is denoted $\Sigma_{\mathcal{M}}$.

Under a given policy π, the resulting set of (infinite) paths has, as for DTMCs, an associated probability measure, which we denote $\Pr^{\pi}_{\mathcal{M}}$. Furthermore, for both memoryless and finite-memory policies, we can build a (finite) *induced DTMC* which is equivalent to $\mathcal{M}$ acting under π.

Rewards. We associate both DTMCs and MDPs with *reward structures*, which are annotations of the states or transitions of a model with numerical values. For consistency with the literature on probabilistic model checking and temporal logics, we use the terminology *rewards* although in practice these can (and often do) represent *costs*, such as time elapsed or energy consumed. For the purposes of our algorithms, we assume that rewards are integer-valued, but we note that these could be defined as rationals, using appropriate scaling.

Definition 5 (Reward structure)**.** A *reward structure* is, for a DTMC $\mathcal{D}$, a function $r : S \to \mathbb{N}$ and, for an MDP $\mathcal{M}$, a function $r : S \times A \to \mathbb{N}$.

For an infinite path ω, we also write $r(\omega, k)$ for the sum of the reward values over the first k steps of the path, i.e., $r(s_0 s_1 s_2 \dots) = \sum_{i=0}^{k-1} r(s_i)$ for a DTMC and $r(s_0 a_0 s_1 a_1 s_2 \dots) = \sum_{i=0}^{k-1} r(s_i, a_i)$ for an MDP. To reason about rewards, we define random variables over the executions (infinite paths) of a model, typically defined as the total reward accumulated along a path, up until some event occurs. Formally, for a DTMC $\mathcal{D}$, such a random variable is defined as a function of the form $X : \mathit{IPaths}_{\mathcal{D}} \to \mathbb{R}$, with respect to the probability measure $\Pr_{\mathcal{D}}$ over $\mathit{IPaths}_{\mathcal{D}}$. For an MDP $\mathcal{M}$ and policy $\pi \in \Sigma_{\mathcal{M}}$, a random variable is defined as a function $X : \mathit{IPaths}_{\mathcal{M}} \to \mathbb{R}$, with respect to the probability measure $\Pr^{\pi}_{\mathcal{M}}$.

3.2.3 Linear Temporal Logic

We use linear temporal logic (LTL) [16] to express safety specifications. In addition to propositional logical operators, LTL employs temporal operators such as `X` (next), `U` (until), `G` (always), and `F` (eventually). The set of words that satisfies an LTL formula ϕ represents a language $\mathcal{L}(\phi) \subseteq (2^{\mathsf{AP}})^{\omega}$, where AP is a given set of atomic propositions. LTL formulas can be used to express a wide variety of requirements.

Definition 6 (Co-safe LTL [87])**.** Formulae in (syntactically) *co-safe LTL*, over a set of atomic propositions AP, are defined by the grammar:

$$\psi \coloneqq \texttt{true} \mid \texttt{a} \mid \neg\texttt{a} \mid \psi \wedge \psi \mid \psi \vee \psi \mid \texttt{X}\,\psi \mid \texttt{F}\,\psi \mid \psi\,\texttt{U}\,\psi$$

where $\texttt{a} \in AP$ is an atomic proposition.

We use the temporal operators $\texttt{X}\,\psi$ (ψ holds from the *next* state), $\texttt{F}\,\psi$ (ψ *eventually* holds) and $\psi_1\,\texttt{U}\,\psi_2$ (ψ_2 eventually holds and ψ_1 holds *until* that point); see, e.g., [88] for the formal semantics. LTL formulae are evaluated over infinite paths of a model labelled with atomic propositions from the set AP but, for use with cumulative reward, we restrict our attention to the *co-safe* fragment, containing formulae which are satisfied in finite time. Formally, this means that any satisfying path ($\omega \models \psi$) has a *good prefix* i.e., a finite path prefix ω' such that $\omega'\omega'' \models \psi$ for any suffix ω''. For simplicity, Definition 6 defines a syntactic subset of LTL (where negation occurs only at the level of atomic propositions and the *globally* operator is omitted) which is guaranteed to be co-safe.

Chapter 4

Safe Multi-Agent Reinforcement Learning

With the rise of cyber-physical systems (CPS) and the age of the Internet of Things (IoT) more multi-agent systems will need to be controlled using advanced techniques [89]. Multi-agent reinforcement learning (MARL) is a group of learning based methods where multiple agents interact with each other in a common environment for several sequential steps. In recent years, MARL methods have been increasingly used in a wide range of safety-critical applications from traffic management [1] to robotic control [2] to autonomous driving [3].

Existing MARL methods [4], [5] focus mostly on optimizing policies based on returns, none of which can guarantee safety (e.g., no unsafe states are ever visited) during the learning process. Nevertheless, learning with provable safety guarantees is necessary for many safety-critical MARL applications where the agents (e.g., robots, autonomous cars) may break during the exploration process and lead to catastrophic outcomes. In this chapter, the objective is to guarantee that no states deemed unsafe are ever visited.

A recent work [15] developed a shielding framework for single-agent reinforcement learning (RL), which synthesizes a shield to enforce the correctness of safety specifications in linear temporal logic (LTL) [16]. The shield guarantees safety during learning by monitoring the RL agent's actions and preventing the exploration of any unsafe action that violates the LTL safety specification. In this chapter, we adapt the shielding framework to the multi-agent setting. Guaranteeing safety for multiple agents with potentially competing

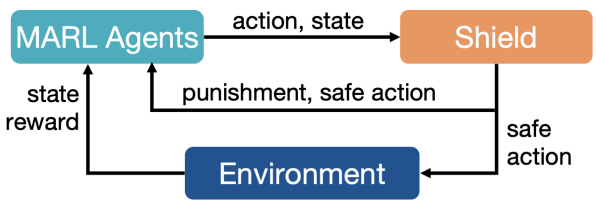


FIGURE 4.1: Overall architecture for shielding in MARL.

goals is more challenging than the single-agent setting, because safety is an emergent property that concerns the coupling of all agents. In addition, the combinatorial nature of MARL (i.e., the joint state space and joint action space increase exponentially with the number of agents) poses scalability issues to the computation of shields. Furthermore, we focus on safety specifications, which are informally interpreted as "something bad should never happen". For example, the LTL formula $\mathtt{G}\ \neg\mathsf{unsafe}$ expresses that "unsafe states should never be visited". An LTL safe specification can be translated into a safe language accepted by a deterministic finite automaton (DFA) [90].

We present in this chapter the first work to provide safety guarantees (expressed as LTL specifications) for MARL. Our contributions are threefold. First, we develop a *centralized shielding* approach for MARL, where we synthesize a single shield to centrally monitor the joint actions of all agents. The shield determines that a joint action is safe if all agents satisfy the safety specification. We follow the *minimal interference* principle proposed in [15]; that is, a shield should restrict the agents as infrequently as possible and only corrects the actions that violate the safety specification. Moreover, we introduce an additional interpretation of minimal interference in the multi-agent setting: a shield should change the actions of as few agents as possible when correcting an unsafe joint action. The centralized shielding approach has limited scalability, because the computational cost of synthesizing shields depends on the number of MARL agents and the complexity of the safety specification.

Second, we develop a *factored shielding* approach for MARL to address the aforementioned scalability issues. The factored shielding offers a divide-and-conquer approach: multiple shields are computed based on a factorization of the joint state space observed by all agents. The set of factored shields monitors agents concurrently and each shield is only responsible for a subset of agents at each step. Agents can join or leave a factored shield at any time depending on their states. Factored shields enforce the correctness of safety specification by preventing unsafe actions similarly to the centralized shield. While each individual factored shield can only monitor a limited number of agents due to the restriction of shield computation, we can employ as many shields as needed; and together the set of factored shields can monitor a large number of MARL agents.

Third, we showcase the performance of the two shielding approaches via experimental evaluation on six benchmark problems in a grid world [91] and a cooperative navigation [92] environment. We used two MARL algorithms, CQ-learning [85] and MADDPG [38], in our experiments to demonstrate that the shielding approaches are compatible with different MARL algorithms. Experimental results show that the shielding approaches can both guarantee the safety of agents during learning without compromising the quality of learned policies; moreover, factored shielding is more scalable in the number of agents than centralized shielding.

4.1 Background

In this section, we go over relevant notation and preliminaries needed for this chapter. Given an alphabet Σ, we denote by Σ^ω and Σ^* the set of infinite and finite words over Σ, respectively.

4.1.1 Safety Specifications and Safety Games

Formally, a *deterministic finite automaton* is a tuple $(Q, q_0, \Sigma, \delta, F)$ with a finite set of states Q, an initial state $q_0 \in Q$, a finite alphabet Σ, the transition function $\delta : Q \times \Sigma \to Q$, and a finite set of accepting states $F \subseteq Q$. Let $q_0\sigma_0 q_1\sigma_1 \cdots \in (Q \times \Sigma)^\omega$ be a run of the DFA. The word $\sigma_0\sigma_1 \ldots$ is in the safety language accepted by the DFA if the run only visits accepting states of the DFA, i.e., $q_i \in F$ for all $i \geq 0$.

We use Mealy machines to represent shields. Formally, a *Mealy machine* is a tuple $(Q, q_0, \Sigma_I, \Sigma_O, \delta, \lambda)$ with a finite set of states Q, an initial state $q_0 \in Q$, finite sets of input alphabet Σ_I and output alphabet Σ_O, the transition function $\delta : Q \times \Sigma_I \to Q$, and the output function $\lambda : Q \times \Sigma_I \to \Sigma_O$. For a given input trace $\sigma_0\sigma_1 \cdots \in \Sigma_I^\omega$, the Mealy machine generates a corresponding output trace $\lambda(q_0, \sigma_0)\lambda(q_1, \sigma_1) \cdots \in \Sigma_O^\omega$ where $q_{i+1} = \delta(q_i, \sigma_i)$ for all $i \geq 0$.

As we will describe later, we synthesize shields by solving two-player safety games. Formally, a *two-player safety game* is a tuple $(G, g_0, \Sigma_1, \Sigma_2, \delta, F)$ with a finite set of game states G, an initial state $g_0 \in G$, finite sets of alphabet Σ_1 and Σ_2 for Player 1 and Player 2 respectively, the transition function $\delta : G \times \Sigma_1 \times \Sigma_2 \to G$, and a set of safe states $F \subseteq G$ defines the *winning condition* such that a play $g_0 g_1 \ldots$ of the game is winning iff $g_i \in F$ for all $i \geq 0$. At each game state $g_i \in G$, Player 1 chooses an action $a_i^1 \in \Sigma_1$, then Player 2 chooses an action $a_i^2 \in \Sigma_2$, and the game moves to the next state $g_{i+1} = \delta(g_i, a_i^1, a_i^2)$. A memoryless strategy for Player 2 is a function $\kappa : G \times \Sigma_1 \to \Sigma_2$. A *winning region* $W \subseteq F$ is the set of states from which there exists a winning strategy (i.e., all plays constructed using the strategy satisfy the winning condition).

4.2 Approach

4.2.1 Centralized Shielding

We introduce a *centralized shield* (i.e., a single shield for all agents) into the traditional MARL process. In the following, we first describe how the centralized shield interacts with the learning agents and the environment to achieve safe MARL, then we present our method for synthesizing the centralized shield.

Algorithm 1: Centralized shielding at time step t

Input : Shield $\mathcal{S}$, MARL agents' joint action $a_t = (a_t^1, \cdots, a_t^n)$ and joint state $s_t = (s_t^1, \cdots, s_t^n)$, a constant punishment cost c

Output: Safe joint action $\bar{a}_t$, punishment ρ_t

$\rho_t \leftarrow 0$

$\bar{a} \leftarrow$ safe action output by the shield $\mathcal{S}$

forall *agent* k *such that* $\bar{a}^k \neq a^k$ **do**

  $\rho_t^k \leftarrow c$

return $\bar{a}_t$, ρ_t

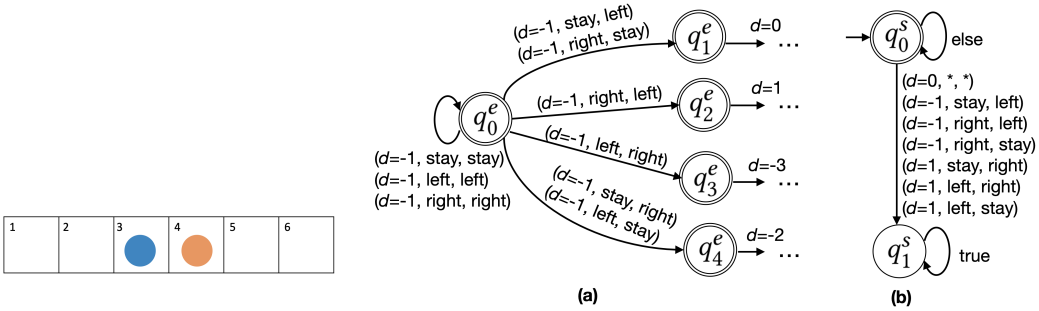
Figure 4.1 illustrates the interaction of the centralized shield, the MARL agents, and the environment. Algorithm 1 summarizes the centralized shield's behavior at time step t. The shield monitors the joint action $a_t = (a_t^1, \cdots, a_t^n)$ chosen by the MARL agents. If the shield detects that a_t is unsafe (i.e., violates the safety specification) at the agents' joint state $s_t \in S$, the shield substitutes a_t with a safe joint action $\bar{a}_t$; otherwise, the shield forwards a_t to the environment directly (i.e., $\bar{a}_t = a_t$). The environment receives the action $\bar{a}_t$ output by the shield, moves to state $s_{t+1} \in S$, and provides reward $R^k(s_t, \bar{a}_t, s_{t+1})$ for each agent k to update its policy. Meanwhile, the shield assigns a punishment ρ_t^k to agent k (where $\bar{a}_t^k \neq a_t^k$) to help the MARL algorithm learn about the cost of unsafe actions.

A centralized shield enforces the safety specification during the learning process (i.e., any unsafe action is corrected to a safe action before being sent to the environment). Moreover, we require the shield to restrict MARL agents as rarely as possible via the *minimal interference* criteria: (1) the shield only corrects the joint action a_t if it violates the safety specification, and (2) the shield seeks a safe joint action $\bar{a}_t$ that changes as few of the agents' actions as possible from a_t.

Our approach synthesizes a centralized shield based on the safety specification and a coarse environment abstraction. Note that we do not require the environment dynamics to be completely known in advance. The shield can be synthesized based on a coarse abstraction of the environment that is sufficient to reason about the potential violations of safety specifications. For example, before deploying a team of robots for a disaster search and rescue mission, we may use some low-resolution satellite imagery to build a coarse, high-level abstraction about the terrain environment for shield synthesis. However, such a coarse environment abstraction is not sufficient for planning algorithms that rely on complete models of the environment. Therefore, MARL agents still need to learn the concrete environment dynamics.

We describe how to synthesize centralized shields as follows. We assume some coarse environment abstraction has been given as a DFA $\mathcal{A}^e = (Q^e, q_0^e, \Sigma^e, \delta^e, F^e)$ with the alphabet $\Sigma^e = L \times A$, where an observation function $f : S \to L$ maps the MARL agents' joint state space S to some observation set L, and A is the joint action set of

all agents. We translate the safety specification expressed as an LTL formula to another DFA $\mathcal{A}^s = (Q^s, q_0^s, \Sigma^s, \delta^s, F^s)$ with the same alphabet $\Sigma^s = L \times A$. We combine $\mathcal{A}^e$ and $\mathcal{A}^s$ into a two-player safety game $\mathcal{G} = (G, g_0, \Sigma_1, \Sigma_2, \delta^g, F)$ where $G = Q^e \times Q^s$, $g_0 = (q_0^e, q_0^s)$, $\Sigma_1 = L$, $\Sigma_2 = A$, $\delta^g((q^e, q^s), l, a) = (\delta^e(q^e, (l \times a)), \delta^s(q^s, (l \times a))$ for all $(q^e, q^s) \in G$, $l \in L$, and $a \in A$, and $F = Q^e \times F^s$. We solve the two-player safety game $\mathcal{G}$ and compute the wining region $W \subseteq F$ using the techniques described in [24]. We construct the centralized shield represented as a Mealy machine $\mathcal{S} = (Q, q_0, \Sigma_I, \Sigma_O, \delta, \lambda)$, where the state space is given by the game states $Q = G = Q^e \times Q^s$, the initial state $q_0 = g_0 = (q_0^e, q_0^s)$, the input alphabet $\Sigma_I = L \times A$, the output alphabet $\Sigma_O = A$; the transition function $\delta(g, (l, a)) = \delta^g(g, l, \lambda(g, (l, a)))$ for all $g \in G$, $l \in L$, and $a \in A$; the output function $\lambda(g, (l, a)) = a$ if $\delta^g(g, l, a) \in W$, and $\lambda(g, (l, a)) = \bar{a}$ if $\delta^g(g, l, a) \notin W$, where $\bar{a} \in A$ is a safe action with $\delta^g(g, l, \bar{a}) \in W$ and only differs from the unsafe action a in terms of the minimal number of agents' actions. We also define a (negative) constant c as punishment for unsafe actions. The computational cost of synthesizing centralized shields grows exponentially as the number of agents increases, and also depends on the complexity of the safety specification and environment abstraction.



(A) (B)

FIGURE 4.2: **Left.** Example grid map with two agents. **Right.** (a) An example environment abstraction DFA $\mathcal{A}^e$. (b) An example safety specification DFA $\mathcal{A}^s$.

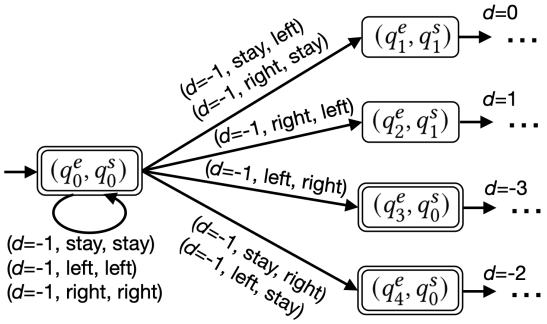


FIGURE 4.3: An example safety game given by the product of $\mathcal{A}^e$ and $\mathcal{A}^s$ shown in Figure 4.2b. Double circles denote safe states.

To exemplify the shield synthesis method, let us consider two agents (blue and orange) in the grid map shown in Figure 4.2a. Each agent can move left or right, or stay in

the same grid. An agent receives a reward of 10 if it reaches grid 1 or 6, and receives a negative reward of -1 if it collides with the other agent. The discount factor being $\gamma = 1$. Each agent tries to learn an optimal policy based on the observed rewards. However, the negative reward cannot completely prevent collisions during the learning process of traditional MARL algorithms. Because the agents need to explore different (even unsafe) actions to learn about states and rewards from the environment. Now we show how to construct a shield that can block unsafe actions and guarantee collision free. We use an observation set L that measures the distance d between blue and orange agents. For example, $d = -1$ for agents' positions shown in Figure 4.2a. We build a coarse environment abstraction DFA $\mathcal{A}^e$ that captures the relation of agents' distances and joint actions. Figure 4.2b(a) shows a fragment of $\mathcal{A}^e$ (double circle denotes accepting states of DFAs, * refers to any action). We can express the safety specification of collision avoidance using the following LTL formula:

$$\mathtt{G}\,\neg\Big((d=0)\vee\big((d=-1)\wedge((stay,left)\vee(right,left)\vee(right,stay))\big)\\ \vee\big((d=1)\wedge((stay,right)\vee(left,right)\vee(left,stay))\big)\Big)$$

which indicates that the following bad scenarios should never occur: two agents being in the same grid ($d = 0$), or taking certain unsafe joint actions that would make them collide into each other when $d = -1$ or $d = 1$. We can translate the LTL formula into the DFA $\mathcal{A}^s$ shown in Figure 4.2b(b). We build a two-player safety game from the product of $\mathcal{A}^e$ and $\mathcal{A}^s$. Figure 4.3 shows a fragment of the safety game (double circles denote safe states). For example, in the game state (q_0^e, q_0^s), the blue and orange agents should not choose a joint action (*stay*, *left*) that leads to an unsafe game state (q_1^e, q_1^s) where two agents collide into each other. The synthesized centralized shield prevents the collision by correcting the unsafe action (*stay*, *left*) with a safe action (*stay*, *stay*) and assigns a punishment cost of -1 to the orange agent.

Correctness. We show that the synthesized centralized shields can indeed enforce safety specifications for MARL agents as follows. Given a trace $s_0 a_0 s_1 a_1 \cdots \in (S \times A)^\omega$ jointly produced by MARL agents, the centralized shield, and the environment, there is a corresponding run $q_0 q_1 \cdots \in Q^\omega$ of the shield $\mathcal{S} = (Q, q_0, \Sigma_I, \Sigma_O, \delta, \lambda)$ such that $q_{i+1} = \delta(q_i, (f(s_i), a_i))$ and $a_i = \lambda(q_i, (f(s_i), a_i))$ for all $i \geq 0$, where $f : S \to L$ is the observation function. By the construction of the shield, we have $Q = Q^e \times Q^s$, where Q^e and Q^s are the state space of the environment abstraction DFA $\mathcal{A}^e$ and the safety specification DFA $\mathcal{A}^s$, respectively. Thus, we can project the run $q_0 q_1 \ldots$ of the shield onto a trace $q_0^s(f(s_0), a_0)q_1^s(f(s_1), a_1)\ldots$ on $\mathcal{A}^s$. The shield is constructed from the winning region of the safety game, which ensures that only safe states are ever visited along the trace $q_0^s(f(s_0), a_0)q_1^s(f(s_1), a_1)\ldots$ of $\mathcal{A}^s$ (i.e., $q_i^s \in F^s$ for all $i \geq 0$). Thus, the centralized shield $\mathcal{S}$ can guarantee that the safety specification $\mathcal{A}^s$ is never violated.

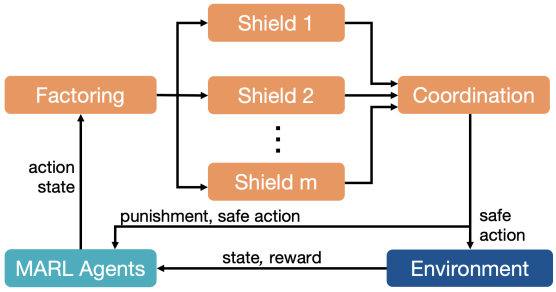


FIGURE 4.4: Safe MARL with factored shielding.

Impact on Learning Performance. The centralized shielding approach is agnostic to the choice of a MARL algorithm, because the shield interacts with the learner only via inputs and outputs, and does not rely on the inner-workings of the learning algorithm. As explained in Section 3.1, there is a lack of theoretical convergence guarantees for MARL algorithms in general. Thus, a full theoretical analysis of the shielding approach's impact on MARL convergence is out of scope for this chapter. We show empirically in our experiments (Section 4.3) that (1) MARL with and without centralized shielding both converge; (2) centralized shielding can guarantee the safety in all examples, while MARL without shielding does not prevent agents' unsafe behavior; (3) centralized shielding learns more optimal policies with better returns than non-shielded MARL in some examples (e.g., due to the removal of unsafe actions that may destabilize learning).

4.2.2 Factored Shielding

The centralized shielding approach has limited scalability, because the computational cost of shield synthesis grows exponentially with the number of agents. To address this limitation, we develop a *factored shielding* approach that synthesizes multiple shields to monitor MARL agents concurrently, as illustrated in Figure 4.4.

Let us consider a finite set of factored shields $\{\mathcal{S}_1, \cdots, \mathcal{S}_m\}$ where each shield is synthesized based on a factorization of the joint state space observed by all agents. We can leverage problem-specific knowledge to achieve an efficient factorization scheme (e.g., how many shields to use, what is the state space covered by each shield). For example, we synthesize two factored shields $\mathcal{S}_1$ and $\mathcal{S}_2$ for monitoring agents' behavior in grids 1-3 and 4-6 of Figure 4.2a, respectively. A factored shield monitors a subset of agent actions at each time step. A shield is not tied to any specific agent; instead, an agent can request to join or leave a shield from border states at any time. For example, if the orange agent in Figure 4.2a wants to move from grid 4 to grid 3, it would request to join $\mathcal{S}_1$ and leave $\mathcal{S}_2$.

Algorithm 2 describes how the factored shielding works at each time step t. There are three phases: (1) factorization, (2) shielding, and (3) coordination. In the factorization

phase (line 5-14), the algorithm identifies the factored shields that are responsible for monitoring each agent k in the current time step t, based on a mapping between the agent state s_t^k and the factored state space assigned to each shield $\mathcal{S}_i$. Thus, there must exist at least one factored shield monitoring each agent. If agent k happens to be in a border state s_t^k within the shield $\mathcal{S}_i$ and, by taking action a_t^k, the agent would cross the border to another shield $\mathcal{S}_j$, the algorithm relates agent k with both shields and renames its actions in shield $\mathcal{S}_i$ and $\mathcal{S}_j$ as *leave* and *join*, respectively. Next, in the shielding phase (line 16-33), each factored shield checks if the set of related agents act safely (i.e., not violating the safety specification within it) and substitutes any unsafe action with a default safe action (e.g., *stay* in our running example). In the coordination phase (line 35-47), the algorithm checks the output of all shields to make sure compatible decisions are made for each agent. For example, if an agent action a_t^k is translated to requests of leaving $\mathcal{S}_i$ and joining $\mathcal{S}_j$, then both requests need to be approved by the shields; however, if $\mathcal{S}_j$ considers *join* as unsafe at this time and substitutes with a default safe action *stay*, then the algorithm corrects the agent action a_t^k and output with safe action $\bar{a}_t^k = stay$, Finally, the algorithm assigns a punishment cost $\rho_t^k = c$ for any unsafe action a_t^k with $\bar{a}_t^k \neq a_t^k$.

We synthesize factored shields using a similar method as the synthesis of centralized shields. However, instead of building a safety game that accounts for the joint states S and joint actions $A = A^1 \times \cdots \times A^n$ of all MARL agents, we only consider a factorization of states and actions for the synthesis of each factored shield. Let $S_i \subseteq S$ be the factored state space of shield $\mathcal{S}_i$. We factor the coarse environment abstraction DFA $\mathcal{A}^e$ into a DFA $\mathcal{A}_i^e = (Q_i^e, q_{0,i}^e, \Sigma_i^e, \delta_i^e, F_i^e)$ with the alphabet $\Sigma_i^e = L_i \times A_i$, where an observation function $f : S_i \to L_i$ maps the factored states S_i to some observation set L_i, and $A_i = (A^1 \cup \cdots \cup A^n \cup \{join, leave\}) \times \cdots \times (A^1 \cup \cdots \cup A^n \cup \{join, leave\})$ is the joint action in shield $\mathcal{S}_i$ with $|A_i|$ determined by the maximum number of agents that shield $\mathcal{S}_i$ can monitor at once. Note that we need to translate the agent actions at border states of a shield to *join* or *leave* requests. Intuitively, since any agent may request to join or leave shield $\mathcal{S}_i$ at any time, the joint action A_i needs to account for any possible combination of agents. This allows us to synthesize factored shields offline with a fixed alphabet, instead of re-computing shields for different agents at each step during learning. Similarly, we can factor the safety specification DFA $\mathcal{A}^s = (Q^s, q_0^s, \Sigma^s, \delta^s, F^s)$ into a DFA $\mathcal{A}_i^s = (Q_i^s, q_{0,i}^s, \Sigma_i^s, \delta_i^s, F_i^s)$ with the alphabet $\Sigma_i^s = L_i \times A_i$. We obtain the shield $\mathcal{S}_i$ as a Mealy machine by solving the two-player safety game $\mathcal{G}_i$ built from $\mathcal{A}_i^e$ and $\mathcal{A}_i^s$, in a similar way as described in Section 4.2.1.

Algorithm 2: Factored shielding at time step t

Input : A set of factored shields $\{\mathcal{S}_1, \cdots, \mathcal{S}_m\}$, MARL agents' joint action $a_t = (a_t^1, \cdots, a_t^n)$ and joint state $s_t = (s_t^1, \cdots, s_t^n)$, a default safe action b, a constant punishment cost c

Output: Safe joint action $\bar{a}_t$, punishment ρ_t

Init int array $\mathsf{A2S} : n \times 2$ // related shield index
Init string array $\mathsf{Act} : n \times 2$ // actions
Init Boolean array $\mathsf{S2A} : m \times n$ // agents in each shield
// Factorization phase
forall *agent* $k \in \{1, \cdots, n\}$ **do**
    find a factored shield $\mathcal{S}_i$ related to the agent state s_t^k
    if (s_t^k, a_t^k) *may leave shield* $\mathcal{S}_i$ *and join shield* $\mathcal{S}_j$ **then**
        $\mathsf{A2S}[k][0] \leftarrow i$, $\mathsf{A2S}[k][1] \leftarrow j$
        $\mathsf{Act}[k][0] \leftarrow$ "*leave*", $\mathsf{Act}[k][1] \leftarrow$ "*join*"
        $\mathsf{S2A}[i][k] \leftarrow \mathsf{True}$, $\mathsf{S2A}[j][k] \leftarrow \mathsf{True}$
    else
        $\mathsf{A2S}[k][0] \leftarrow i$, $\mathsf{Act}[k][0] \leftarrow a_t^k$, $\mathsf{S2A}[i][k] \leftarrow \mathsf{True}$
// Shielding phase
forall *shield* $\mathcal{S}_i$ *with* $i \in \{1, \cdots, m\}$ **do**
    $a \leftarrow \{\}$
    forall k *with* $\mathsf{S2A}[i][k] = \mathsf{True}$ **do**
        if $\mathsf{A2S}[k][0] = i$ **then**
            $a \leftarrow$ append $\mathsf{Act}[k][0]$
        else
            $a \leftarrow$ append $\mathsf{Act}[k][1]$
    $\bar{a} \leftarrow$ safe action output by the shield $\mathcal{S}_i$
    forall *agent* k *such that* $\bar{a}^k \neq a^k$ **do**
        if $\mathsf{A2S}[k][0] = i$ **then**
            $\mathsf{Act}[k][0] \leftarrow \bar{a}^k$
        else
            $\mathsf{Act}[k][1] \leftarrow \bar{a}^k$
// Coordination
forall *agent* $k \in \{1, \cdots, n\}$ **do**
    if $\mathsf{A2S}[k][1] \neq null$ **then**
        if $\mathsf{Act}[k][0] =$ "*leave*" *and* $\mathsf{Act}[k][1] =$ "*join*" **then**
            $\mathsf{Act}[k][0] \leftarrow a_t^k$
        else
            $\mathsf{Act}[k][0] \leftarrow b$
    $\bar{a}_t^k \leftarrow \mathsf{Act}[k][0]$, $\rho_t^k \leftarrow 0$
    if $\bar{a}_t^k \neq a_t^k$ **then**
        $\rho_t^k \leftarrow c$
return $\bar{a}_t$, ρ_t

Figure 4.5 shows an example safety game for synthesizing the shield $\mathcal{S}_1$ that monitors agents' actions in grid 1-3 of our running example. To simplify the graphic notation, we put observations inside each state which should be labeled on all outgoing transitions from that state. The observations are about agents' grid positions, with ∞ denoting outside. $*$ refers to any action except "join". The initial game state observes that the blue agent is in grid 3 and the orange agent is outside the shield. If the blue and orange agents ask for a pair of actions (*stay*, *join*), then the game would move to an unsafe state where both agents collide into each other in grid 3. In this case, shield $\mathcal{S}_1$ substitutes (*stay*, *join*) with safe actions (*stay*, *stay*). Since the orange agent is involved in two shields $\mathcal{S}_1$ and $\mathcal{S}_2$, we need to coordinate the output of both shields. For example, if $\mathcal{S}_1$ rejects orange agent's *join* request but $\mathcal{S}_2$ accepts the same agent's *leave* request, then there is conflict among the output of $\mathcal{S}_1$ and $\mathcal{S}_2$. In such case, our coordination algorithm chooses the default safe action *stay* for the orange agent. Note that, if there is another agent in shield $\mathcal{S}_2$, then it should not be allowed to move to grid 4 before the orange agent successfully leaves $\mathcal{S}_2$ to avoid collision. Such safety constraints can be encoded in the safety game for synthesizing the shield $\mathcal{S}_2$.

Correctness. We show that the factored shielding algorithm can guarantee safety for MARL agents. Given a trace $s_0 a_0 s_1 a_1 \cdots \in (S \times A)^\omega$ jointly produced by MARL agents, the factored shielding, and the environment, we prove that the state-action pair (s_t, a_t) is safe at every time step t. There are several cases. First, suppose none of the agents requests to switch shields at time step t. By the construction of factored shields, each shield $\mathcal{S}_i$ monitors a subset of agents based on the factored state space $s_{t,i}$ and outputs a safe joint action $a_{t,i}$ that does not violate the safety specification. Thus, the joint state $s_t = s_{t,1} \cup \cdots \cup s_{t,m}$ and joint action $a_t = a_{t,1} \cup \cdots \cup a_{t,m}$ output by all shields are safe for all agents. Second, suppose there is some agent k requesting to leave shield $\mathcal{S}_i$ and join shield $\mathcal{S}_j$. If both shields accept agent k's requests, which means that agent k does not cause a violation of safety specification with either shield. So we still have s_t and a_t safe for all agents. If $\mathcal{S}_j$ rejects agent k's joining request and substitutes with a default

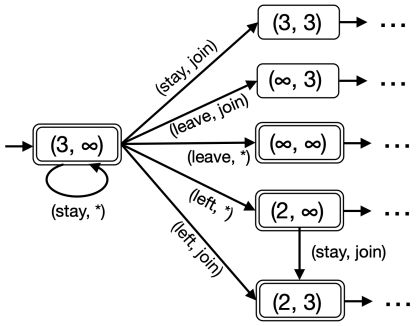


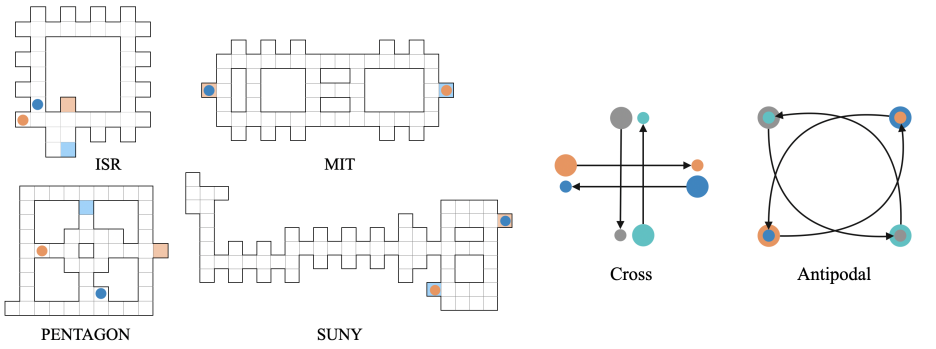
FIGURE 4.5: An excerpt of the safety game for constructing shield $\mathcal{S}_1$ of our running example. Double lines indicate safe states.

safe action, then the factored shielding algorithm coordinates with the output of $\mathcal{S}_i$ and corrects agent k's leaving request with the default safe action as well. Such a correction does not affect the safety of other agents in shield $\mathcal{S}_i$, because by construction the shield accounts for the worst case scenario of leaving request being rejected. Therefore, we have the joint state-action pair (s_t, a_t) safe at every time step t for all agents.

Impact on Learning Performance. Similarly to centralized shielding, the factored shielding approach is agnostic to the choice of a MARL algorithm. We show empirically via our experiments that adding factored shields does not prevent MARL algorithms from converging. In addition, our experiments show that the factored shielding approach can be applied to examples where the synthesis of centralized shields is not feasible due to a large number of agents. While the two shielding approaches can both guarantee the safety during learning in all examples, factored shielding sometimes leads to less optimal policies than centralized shielding (e.g., due to the delay caused by agents switching shields).

4.3 Experiments

We implemented both the centralized shielding and factored shielding approaches in Python and used the Slugs tool [93] to synthesize shields via solving two-player safety games. We applied our prototype implementation to six benchmark problems in the grid world (Figure 4.6a) and a cooperative navigation environment (Figure 4.6b). We used two MARL algorithms CQ-learning [85] and MADDPG [38] in experiments to show that our shielding approaches are agnostic to the choice of MARL algorithms. The experiments were run on a computer with Intel i5 CPU and 16 GB of RAM. Each experiment was split into training phase (linearly decreasing exploration) and evaluation phase (immediately following the training phase and with an exploration rate of 5%). All experiments were conducted for 10 independent runs whose results were averaged to reduce the impact of outliers. The shields in all examples were synthesized within two minutes.



(A) Grid world maps [91]. (B) Cooperative navigation maps [92].

FIGURE 4.6: Map visualizations

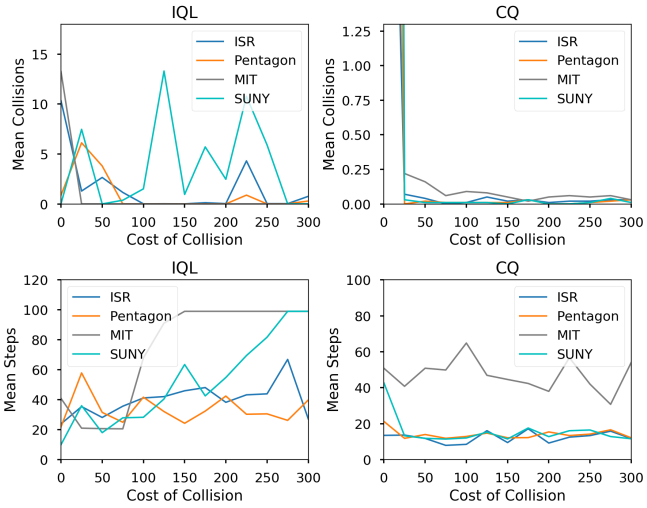


FIGURE 4.7: Collision variation experiments results.

Problem Setup. Figure 4.6a shows four maps of benchmark grid world examples adapted from [91]. Each map has two agents (blue and orange), where each agent aims to learn its own optimal policy for navigating from the start position (circles) to the target position (squares) while trying to avoid collisions. Each agent has five possible actions: *stay*, *up*, *down*, *left*, *right*. Once an agent reaches its target position, it stays there. A learning episode ends when both agents have reached their target positions. Both agents have the same reward function: -1 for a valid move, -10 for a collision with a wall, -30 for collision with the other agent, 100 for arriving at the agent's target position.

Figure 4.6b shows two benchmark cooperative navigation examples adapted from [92]. Each example has four agents (blue, orange, green, and grey) represented as particles. Their start positions are shown as large circles. The goal is for agents to cooperate and reach their designated target (small circles) positions as fast as possible while avoiding collisions. We discretize the fully continuous environment in [92] by restricting agents only take positions with a precision of 0.1. An agent receives a higher reward when it gets closer to its target position (i.e., negation of the distance value), and a negative reward -1 for any collision.

Collision Variation Experiments. We conducted a set of experiments using the grid world examples to highlight why relying on the reward function only is not sufficient to achieve safety (i.e., collision avoidance in our examples). To prevent collisions, the traditional practice of reinforcement learning is to assign a negative reward (we refer to its absolute value as the cost of collision) whenever a collision occurs, and increase the cost until the probability of collision happening becomes negligible. Figure 4.7 shows the results (avg. of 10 evaluation episodes conducted after 1,000 training episodes, across

		IQL			CQ			CQ with centralized shield			CQ with factored shield		
Maps	Opt. Steps	Steps	Reward	Coll.	Steps	Reward	Coll.	Steps	Reward	Coll.	Steps	Reward	Coll.
ISR	5	30.35	-10.20	20.30	8.66	89.53	0.40	7.03	93.85	**0.00**	7.31	93.74	**0.00**
Pentagon	10	46.58	-19.17	11.60	10.96	88.96	0.20	12.08	88.44	**0.00**	13.20	84.88	**0.00**
MIT	18	20.84	77.33	0.00	42.93	30.38	0.90	28.38	73.94	**0.00**	29.96	37.96	**0.00**
SUNY	10	34.80	-160.175	72.60	13.97	84.78	0.30	11.97	88.44	**0.00**	14.02	83.77	**0.00**

TABLE 4.1: Results comparing the independent Q-learning, CQ-learning with and without shields (mean of 10 evaluation episodes conducted after 1,000 training episodes, across 10 independent runs).

10 independent runs) of our experiments using the independent Q-learning [84] and CQ-learning[85]. The left side of the figure shows that, for the independent Q-learning, increasing the cost of collision cannot guarantee that the evaluation phase will be completely collision free; moreover, the increased cost of collision leads to a significant agent performance degradation measured by a larger number of steps to reach target positions. In the MIT and SUNY maps, agents even learn policies that give up the primary task of reaching target positions in order to avoid the high collision cost. The results of the CQ-learning (shown in the right side of the figure) are better than those of the independent Q-learning. The number of collisions drops quickly with a relatively low cost. However, CQ-learning cannot guarantee zero collision either (see Table 4.1).

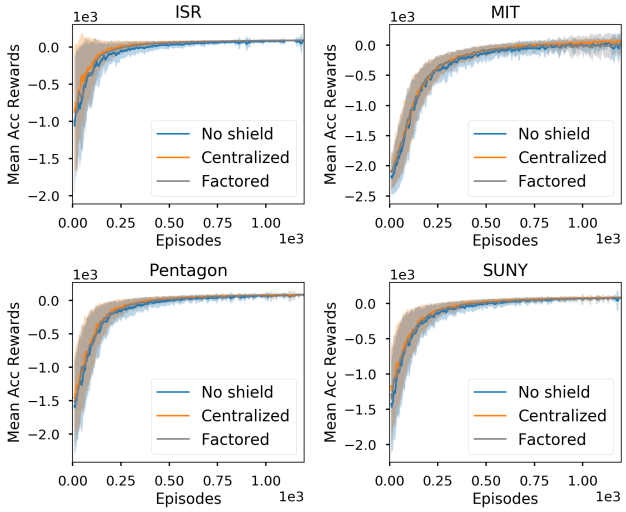


FIGURE 4.8: Comparison of CQ-learning without shielding, with centralized or factored shielding (1,000 training episodes, across 10 runs).

Centralized Shielding Evaluation. We integrated CQ-learning with centralized shielding and applied it to the four grid world examples shown in Figure 4.6a. The results in Table 4.1 show that centralized shielding can guarantee collision free learning in all cases. Moreover, in three out of four maps, CQ-learning with centralized shield obtained better policies with higher rewards and smaller number of steps to reach the target, compared to no shielding. Figure 4.8 shows that centralized shielding achieves the highest accumulated reward in most times; moreover, the blue shaded area (standard deviation of no shielding)

tends to stretch lower than others, indicating that CQ-learning without shielding obtains lower rewards than with centralized shielding on average. The learning curves also show that the centralized shielding does not prevent the learner from converging across different examples. However, we failed to synthesize centralized shields with more than two agents in these grid maps, due to scalability issues of shield synthesis.

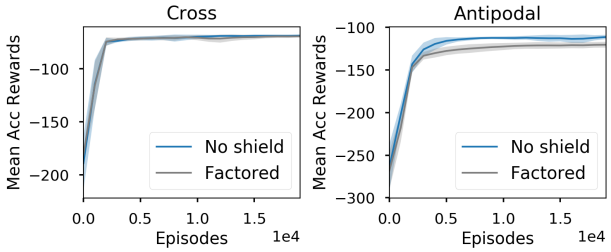


FIGURE 4.9: Comparison of MADDPG without and with factored shielding (20,000 training episodes, across 10 independent runs).

	MADDPG	MADDPG with Shield
Cross	207.20	**0.00**
Antipodal	14,419.20	**0.00**

TABLE 4.2: Total number of collisions over 20,000 training episodes for the cooperative navigation examples.

Factored Shielding Evaluation. First, we applied CQ-learning with factored shielding to the four grid world examples. We adopted a factorization scheme such that each shield monitors agent actions occurring within a 3×3 grid block in each map. Results in Table 4.1 show that CQ-learning with factored shielding can guarantee zero collisions in all examples, while learned policies have similar quality as those obtained from CQ-learning with centralized shielding. Figure 4.8 shows that factored shielding achieves similar performance in terms of the accumulated rewards per episode, compared to centralized shielding and without shielding. Due to the scalability limitation of CQ-learning, we can only consider two agents in these examples.

Additionally, we integrated a different algorithm MADDPG [38] with factored shielding and applied it to the cooperative navigation examples shown in Figure 4.6b with a 5×5 shield size where one unit of distance corresponds to 0.1 in the environment. There are four agents in each example, which is not feasible for centralized shielding approach to handle. Table 4.2 shows that MADDPG with factored shielding can guarantee zero collisions over the training period of $20,000$ episodes for both examples. By contrast, MADDPG without shielding leads to about 207 and $14,419$ occurrences of collisions for the cross and antipodal examples, respectively. Figure 4.9 shows that in the cross example, MADDPG without and with factored shielding have comparable learning performance in terms of the accumulated rewards per episode; in the antipodal example, MADDPG without shielding achieves higher rewards than MADDPG with factored shielding, though

this comes at a trade-off of more collisions. The learning curves in Figure 4.9 also show that the factored shielding do not have negative impact on the learner's ability to converge.

Summary. Our experiments demonstrate that the two shielding approaches can guarantee the safety, without compromising the learning performance in terms of the convergence rate and the quality of learned policies. Moreover, factored shielding is more scalable in the number of agents than centralized shielding.

4.4 Summary

In this chapter, we present two shielding approaches that guarantee the safety specifications expressed in linear temporal logic (LTL) during the learning process of MARL. The centralized shielding approach synthesizes a single shield to centrally monitor the joint actions of all agents and only corrects any unsafe action that violates the LTL safety specification. However, the scalability of centralized shielding is restricted because the computational cost of shield synthesis grows exponentially with the number of agents. The factored shielding approach addresses this limitation by synthesizing multiple factored shields with each shield monitoring a subset of agents at each time step. Our experimental results show that both shielding approaches can guarantee the safety specification (e.g., collision avoidance) during learning, and achieve similar learning performance (e.g., convergence speed, quality of learned policies) as non-shielded MARL.

One of the downsides is that we manually devise factorization schemes for the factored shielding approach in our experiments based on problem-specific knowledge. A possible solution would be to use learning to have adaptive and efficient factorization schemes. However, learning a factorization still needs to maintain our safety objectives. Further investigation of this idea is necessary to determine its feasibility. The methods presented here is such that the shield is guaranteed at all times to monitor the actions. While our approach attempts to guide the learning as well as block unsafe behavior, there is insufficient evidence to suggest that the policy learned would be safe without the shield. Recent work for shielding in partially observable environments looks at ways to gradually remove the shield [94]. Additionally, the abstraction used to synthesized the shield limits the amount of non-determinism and probabilistic that can be present in the environment. Currently, the shield cannot recover from unexpected violations. In the next chapter, we present an approach where we directly use the logic specification to fashion the reward function to guide the learning process. We explore the feasibility of using logic-based reward shaping as an alternative to shielding.

Chapter 5

Logic-Guided MARL Reward Shaping

In the previous chapter, we investigated how to synthesize shields preventing unsafe behavior and how to integrate them in the learning process. However, the creation of the shields relies on an MDP abstraction of certain aspects of the environment. In some cases, not even an abstraction is available. In this chapter, we seek to answer the following research question. How can we better guide MARL agents towards safe policies when the underlying MDP is completely unknown?

While MARL algorithms themselves are appealing for their capability to adapt and learn in an unknown environment, this is only possible if the reward function is designed correctly for the problem the agent is attempting to solve. Designing the reward function is still mostly done manually in most cases and is often hard to interpret [14], [30]. For this reason, logic-based reward shaping is a promising technique which uses the flexibility of Linear Temporal Logic (LTL) to automate the reward function design process based on the task. LTL has also proven to be an expressive and widely used task specification language in robotic applications as well as a possible candidate for translation from natural language [95]. Furthermore, we believe that this could be very useful in the multi-agent case where it becomes even more tricky to design the reward function. In MARL, agents' interactions can affect each other's rewards and make it harder to train. In addition, the joint state space and joint action space increase exponentially with the number of agents which poses scalability issues.

Logic-based reward shaping with single agent RL demonstrates efficient task learning as well as flexible task expression thanks to LTL [18]. Once a Limit-Deterministic Büchi Automaton (LDBA) is synthesized based on the LTL task, we can obtain a product Markov Decision Process (MDP) based on the environment MDP and the LDBA automaton.

Then, we can use the accepting states to reward the agent such that it visits the accepting states infinitely often [21], [22]. With an automated method to synthesize the product MDP, the reward function only depends on the product MDP state.

To the best of our knowledge there is only one previous work attempting to use logic-based reward shaping methods in Multi-Agent Systems, which is based on a more limited fragment of LTL and addresses only tasks that can be fully decomposed into individual tasks [33]. Furthermore, MARL is notoriously difficult to train because of the interactions between agents and logic-based reward shaping methods in the single-agent context is unlikely to work in this setting [35].

In this chapter, we present a semi-centralized approach for reward shaping in multi-agent systems, where we synthesize a centralized LDBA which monitors the agents' progress with respect to a given LTL formula. When a set of observed labels violate the LDBA specification, the automaton transitions to a trap state and returns a large negative reward. If the set of observed labels or selected epsilon-actions transition the LDBA to an accepting state, the agents receives a positive reward. We follow the joint rewards mechanism for this approach; that is, all agents receive the same rewards based on the common LDBA state.

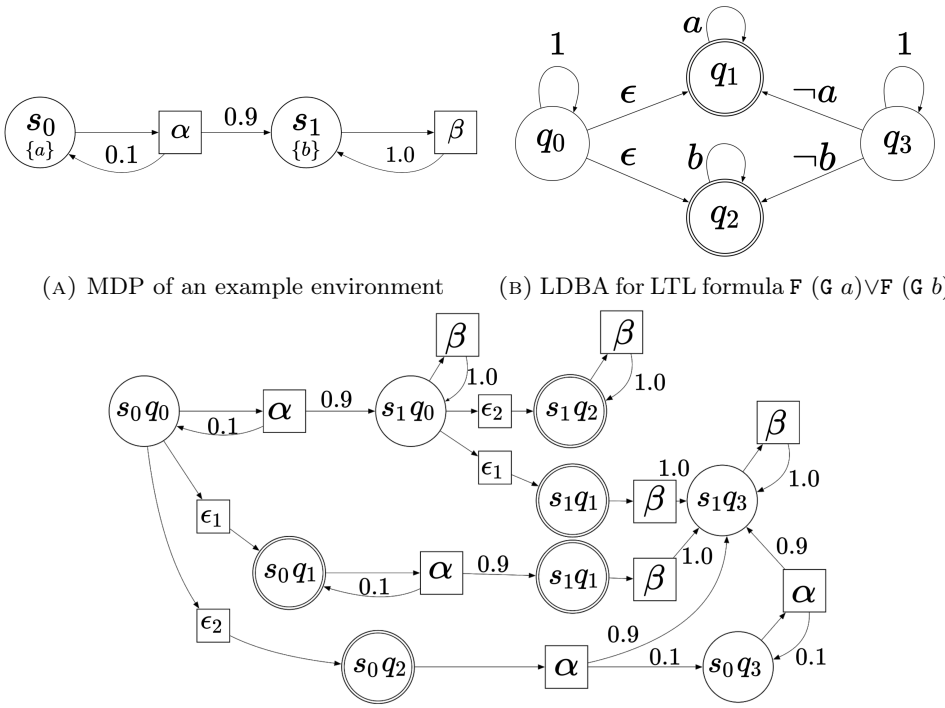
This chapter addresses the need for an automated framework converting tasks to reward functions for their respective agents or a team of agents. Inspired by existing logic-based reward shaping mechanisms for single agent RL, we develop a novel framework for logic-based reward shaping for MARL. Our contributions include:

- We develop a semi-centralized approach for multi-agent reward shaping which is scalable in the number of agents.
- We showcase our approach via experimental evaluation on multiple benchmarks.

Relevant prior work is discussed in section 2.2, then we present important background knowledge with respect to reward shaping in RL in section 5.1. The rest of this chapter is structured as follows. We identify a motivating example and problem statement in section 5.2. In section 5.3, we explain our proposed method in section. We proceed to evaluate our method and present the relevant results in section 5.4. Finally, we discuss limitations in section 5.5 before concluding in section 5.6.

5.1 Background

In figure 5.1, is an example taken from [21] showing the product MDP construction steps. In figure 5.1b, we see the representation of a Limit Deterministic Büchi Automaton (LDBA) which can be automatically generated based on an LTL formula. The main difference compared to a DFA is that the accepting states and the other states are two

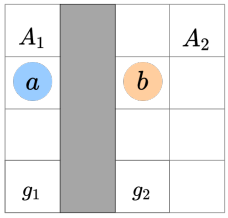


(A) MDP of an example environment (B) LDBA for LTL formula F (G a)∨F (G b)

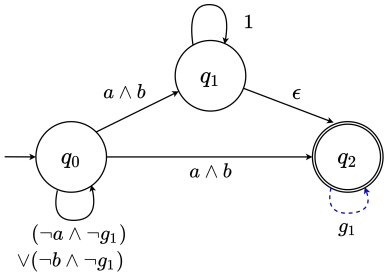
(C) Resulting product MDP

FIGURE 5.1: Constructing the Logic-based Structure

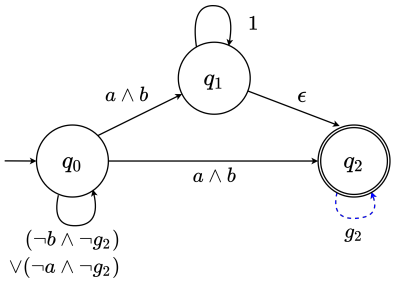
disjoint sets separated by non-deterministic ϵ-transitions [22]. In this figure, the formula intuitively means: "Eventually Always a Or Eventually Always b". In other words, eventually we must only find "a" consistently or only "b" consistently. Here "a" and "b" refer to labels of certain states in the MDP, they could be alternate goal locations for example. Thus, the formula would be encouraging the agent to find one of the goal locations and remain there. Therefore the only accepting states are $\{q_1, q_2\}$, since once you find a state labeled "b" you have to stick to that label otherwise you violate specification and same goes for q_1 with "a". Here q_3 is a failure state from which you cannot recover because of the nature of the specification. Figure 5.1c shows how the LDBA from Figure 5.1b and the MDP from figure 5.1a can be combined into a product MDP. Each state of the new MDP will correspond to a state in the LDBA and a state of the MDP based on the transitions that have been taken and the labels that have been encountered. Moreover, each ϵ-transition in the LDBA gets translated into an action that does not change the MDP state in the product MDP, only the LDBA state. From state s_0q_0 the possible actions are α or the ϵ-transitions (because of q_0). In this MDP, the accepting states are $\{s_0q_1, s_0q_2, s_1q_1, s_1q_2\}$ and the states $\{s_0q_3, s_1q_3\}$ form a non-accepting sink component. In the context of learning, the ϵ-actions can be seen as a guess which may or may not be taken at the right time, in which case they may lead to a non-accepting sink component. This is the case if action ϵ_2 is taken at state s_0q_0. However, in the following episodes, the



(A) Grid representation of the example MDP



(B) Automaton for agent A_1, $\phi_1 = (\neg a \ \mathtt{U} \ b) \wedge \mathtt{F} \ \mathtt{G} \ g_1$



(C) Automaton for agent A_2, $\phi_2 = (\neg b \ \mathtt{U} \ a) \wedge \mathtt{F} \ \mathtt{G} \ g_2$

FIGURE 5.2: Motivating Example

agent should learn that taking action ϵ_2 from state $s_1 q_0$ leads to an accepting strongly connected component (SCC).

The main objective for logic-based reward shaping is to encourage the RL agent to learn a control policy that maximizes the probability of satisfying the Büchi condition for the LTL specification in an arbitrary MDP [22]. The Büchi condition is satisfied when the accepting states of the LDBA automaton are visited infinitely often by the agent's policy. They optimize for this behavior by defining the reward function such that the value of each state in the set of accepting states approaches 1. If an agent visits an accepting state, the agent receives a reward $(1 - \gamma_B)$ and uses an accepting state specific discount value γ_B. If the agents visits a non-accepting state, the agent receives a reward of 0 with a discount value of γ. The probability of satisfying the Büchi condition is maximized as the discount factor γ approaches 1^-. In other work, encouraging the agent to visit the accepting states infinitely is translated into different types of reward functions [21], [27]. In some cases the reward function completely replaces the environment reward and in others balancing factors are used to quantify the relative weight of the different reward sources [21], [30].

5.2 Motivating example

In this work, we first introduce a two agent scenario adapted from an example presented in [33] to motivate our approach. In figure 5.2, agent A_1 needs agent A_2 to press button b

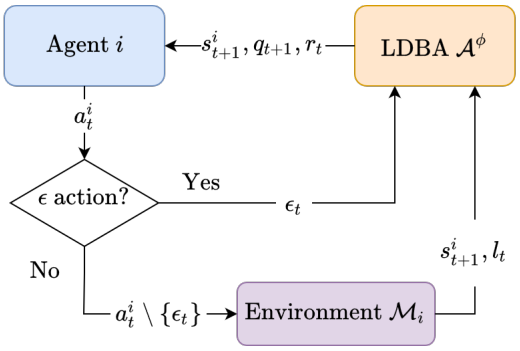


FIGURE 5.3: Semi-centralized Reward Shaping

before proceeding towards its goal g_1 and vice versa. Effectively, both agents need to be pressing the button at the same time before they can reach their goals. In this example, the environment does not enforce this constraint, only that the agents may not pass the wall delineated in gray.

The example in figure 5.2 illustrates the kind of problem which cannot be resolved with a simple independent learners strategy because an agent needs to be aware of labels triggered by other agents. In this chapter, we demonstrate that this can be solved without explicitly splitting the task into multiple individual sub-tasks and use the MARL to learn the sub-tasks.

We consider the problem of learning agent policies in a stochastic multi-agent environment satisfying a desired specification. The environment is modeled as an MDP which may be an abstraction of the actual environment but should be completely characterized for each agent, i.e., one agent's action and state will not affect the other agents actions and MDP states. The desired objective is given by an LTL formula which specifies all requirements to be met and if appropriate in which order. Our goal is for agents to learn policies such that all tasks and requirements are satisfied independently of the underlying MARL algorithm $\mathbb{A}$.

5.3 Approach: Semi-centralized Reward Shaping

In this section, we introduce a *centralized automaton* $\mathcal{A}^\phi$ in the MARL learning process. In the following, we explain how our method can be used to solve problems requiring agent coordination.

Figure 5.3 illustrates the interaction between each agent, the LDBA $\mathcal{A}^\phi$ synthesized from the LTL specification ϕ and the environment represented as an MDP $\mathcal{M}$ from each agent's point of view. At time step t, agent i chooses some action a^i_t based on the underlying algorithm $\mathbb{A}$. Then if the action is an ϵ-action, the epsilon action is forwarded to $\mathcal{A}^\phi$ to

Algorithm 3: Semi-centralized reward shaping at time step t for an agent i

Input : MDP $\mathcal{M}_i$, MARL algorithm $\mathbb{A}$, automaton $\mathcal{A}^\phi$ and augmented state $\langle s_t^i, q_t \rangle$

$\epsilon_t = \emptyset$
foreach *agent* i **do**
  $a_t^i = \mathbb{A}.\texttt{action\_selection}(\langle s_t^i, q_t \rangle, \mathcal{A}_t^i)$ `// Check for epsilon actions`
  if $a_t^i \in \mathcal{E}$ **then**
    $\epsilon_t.\texttt{append}(a_t^i)$
  $s_{t+1}^i = \mathcal{M}_i.\texttt{environment\_step}(s_t^i \setminus \{q_t\}, a_t^i \setminus \{\epsilon_t\})$
$l_t = \texttt{get\_global\_labels}(s_{t+1})$
if $\epsilon_t \neq \emptyset$ **then**
  $q_{t+1}, r_t = \mathcal{A}^\phi.\texttt{update\_automaton}(q_t, l_t, \epsilon_t)$ `// Update automaton and get reward`
$\mathbb{A}.\texttt{update}(\langle s_t, q_t \rangle, \langle s_{t+1}, q_{t+1} \rangle, r_t, a_t)$

update the automaton state. If it is not an ϵ-action then it is sent along with the actions of the other agents to the environment. From the environment, we receive the new set of states and a set of labels. The labeling function need not be part of the environment but can be determined as part of the current states, actions and new states. The labels allow us to transition to the next automaton states (i.e., q_{t+1}) in the LDBA $\mathcal{A}^\phi$ for non ϵ-transitions. In this method, the agents' states are augmented with the automaton state q_t at time t and q_t is identical for each agent. In the case of decentralized execution, each agent may have a copy of the LDBA $\mathcal{A}^\phi$ for which the state q_t is synced at every time step. The purpose of the automaton in this approach is to identify the different states of progress for an LTL specification which can happen regardless of the number of agents involved while giving agents flexibility in the way that the tasks are assigned. Without syncing the automaton states with the other agents, the product MDP for any one agent is incomplete and the agents cannot reach the accepting states.

In algorithm 3, we demonstrate how to implement this method with a generic MARL algorithm $\mathbb{A}$. Instead of explicitly building product a MDP for each agent, we can track and synchronize the progress of each agent in the LDBA. We remark that an agent's state is an augmented state composed of the original MDP state and the automaton state. We first choose an action for each agent i from the available actions at state s_t^i which include ϵ actions if ϵ actions are available at state q_t of the automaton. The environment represented here by MDP $\mathcal{M}_i$ from the point of view of an agent i transitions based on the selected non-ϵ actions. The labels for the new state are then retrieved to transition the automaton for the label-based transitions. For regular transitions, the automaton progresses based on a joint set of labels l_t which corresponds to the union of labels observed by all the agents at time t. For ϵ-action based transitions, only one ϵ-action is allowed per time step in addition to label based transitions where ϵ-actions are resolved before label-based transitions. This is consistent with the definition of an LDBA for which the ϵ-moves can

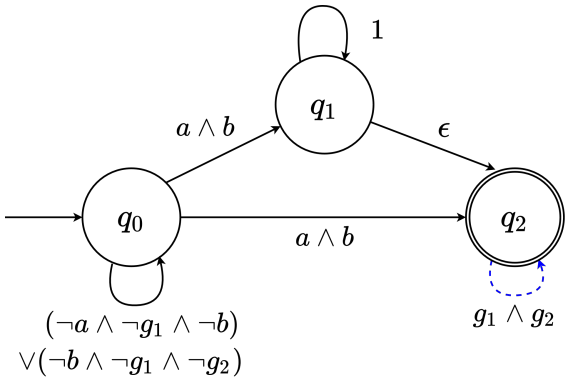


FIGURE 5.4: Automaton Representing Combined LTL Tasks

be taken at any time as long as they are available at the current state of $\mathcal{A}^{\phi}$ [22]. Finally, the MARL algorithm $\mathbb{A}$ is updated with the augmented state and reward received from reward shaping. Using one LDBA in combination with not explicitly building a product MDP enables us to sidestep the issue of exponential increase in the number of product MDP states proportional to the number of agents n.

In the motivating example (fig.5.2), the first agent's automaton would check if button b is triggered to be able to pass towards goal g_1. The second agent's automaton would include button a that can only be reached by the first agent. The respective LTL formulas can be expressed as $\phi_1 = (\neg a \ \mathtt{U} \ b) \wedge \mathtt{F} \ \mathtt{G} \ g1$ and $\phi_2 = (\neg b \ \mathtt{U} \ a) \wedge \mathtt{F} \ \mathtt{G} \ g_2$. In the case of the running example from fig.5.2, we can now express the tasks as follows:

$$\begin{aligned}\phi_3 =&((\mathtt{F} \ a \wedge \neg g_1) \ \mathtt{U} \ (a \wedge b)) \\ &\wedge((\mathtt{F} \ b \wedge \neg g_2) \ \mathtt{U} \ (a \wedge b)) \\ &\wedge(\mathtt{F} \ \mathtt{G} \ g_1) \wedge (\mathtt{F} \ \mathtt{G} \ g_2)\end{aligned}$$

The first line ensures that agents do not attempt to go to g_1 without the buttons being cleared, the second ensures that the agents do not attempt to go to g_2 without the buttons being triggered and the last last line encourages the agents to eventually find the goals g_1 and g_2. The resulting automaton is displayed in fig. 5.4. Notice that the transition from state q_1 to state q_2 requires an epsilon-action. This example demonstrates how using a common automaton can give us flexibility in the the task assignment. Here the user does not need to know which agent should carry out which tasks. The user can still prevent the agents from doing tasks out of order and effectively synchronize the agents. If the buttons represent doorbells and the goals represent opening the doors, the simulation environment could allow the agent to open doors without ringing the doorbells first which can be an undesirable behavior. Furthermore, this example shows that a combined specification ϕ_3 does not necessarily result in a large increase in the number of automaton states (fig.5.4).

Complexity. In terms of time complexity, the main added complexity is the synthesis of the LDBA $\mathcal{A}^\phi$ which can be computed once at the beginning and sent to all agents. In terms of space complexity, we avoid exponential complexity in the number of agents by considering the point of view of only one agent at a time. However, instead of the memory required by agent i being a product of $\mathcal{M}_i$'s state space and $\mathcal{M}_i$'s action space, we now have both an augmented state space and an augmented action space for each agent. However, we argue that although more complex tasks may require a larger number of automaton states, the LDBA of the joint specification may exploit symmetry within the tasks (fig. 5.4). In terms of communication, our method only requires that the value of the automaton state and selection of ϵ-actions be transmitted which is why our method can be considered semi-centralized.

Correctness. We show that the reward function obtained is correct with respect to the joint LTL specification ϕ. Using an LDBA allows us to identify all the different traces through which the joint LTL specification for the MARL tasks may be satisfied. Thus, assuming the labels are identified correctly, following the reward function obtained from the corresponding LDBA is guaranteed to be an accurate representation of the agents' progress. In other words, if the agents receive a non-zero reward then it is guaranteed to correspond to an accepting transition of $\mathcal{M}^x$. We further show that our method is equivalent to building a product MDP $\mathcal{M}^x = \left(\mathcal{A}^\phi \times \mathcal{M}_0 \times \ldots \times \mathcal{M}_n\right)$, $\forall n$ agents after resolving the ϵ-transitions. In our algorithm, we implicitly build the product pairs $\left(\mathcal{A}^\phi \times \mathcal{M}_i\right), \forall i \in n$ where each agent i is responsible for one pairing and the automaton $\mathcal{A}^\phi$ is synchronized.

By definition of a pair product MDP, the actions allowed at a specific state are given by union of the actions allowed in the MDP state and the ϵ-actions allowed in the automaton state. So if we consider the larger picture with all the agents, then the allowed actions at any given state of the product MDP is the union of the action space all pairs for that state which corresponds to the union of the automaton actions with the MDP action space of all agent MDPs. In our case here, the ϵ-actions are not repeated since in the case of multiple ϵ -action only one is selected.

The state space of the pair product MDPs is defined by all the possible combinations of the state space of $\mathcal{A}^\phi$ and the the state space of $\mathcal{M}_i$. If we extend that to all agents, the joint state space is characterized by $\left(\mathcal{A}^\phi \times \mathcal{M}_0\right) \times \ldots \times \left(\mathcal{A}^\phi \times \mathcal{M}_n\right)$. However, since the LDBA state across all agent pair product MDPs is the same, this reduces to $\left(\mathcal{A}^\phi \times \mathcal{M}_0 \times \ldots \times \mathcal{M}_n\right)$, proving it is identical to the product MDP $\mathcal{M}^x$ state space.

The transition function of the product MDP $\mathcal{M}^x$ is defined by the product of the automaton transitions and the successive MDP transitions. If labels resulting in an automaton state transition are identified, the transition of the automaton and the transition of the

MDP state can be combined (fig. 5.1). Because each MDP $\mathcal{M}_i$ transition is fully characterized by one agent, the order of the agent environment transitions during a time step t does not matter (allowing us to take them simultaneously). Therefore, since the set of observed labels l_t at time t is identical for all agents and only one common ϵ action can be taken, the transition computed by each agent for the automaton is deterministic and unique. Thus, the product MDP $\mathcal{M}^x$ transition function reduces to the pair product MDP transitions.

This proves that all aspects of the product MDP $\mathcal{M}^x$ are consistent with our implementation in algorithm 3 of the pair product MDPs for each agent.

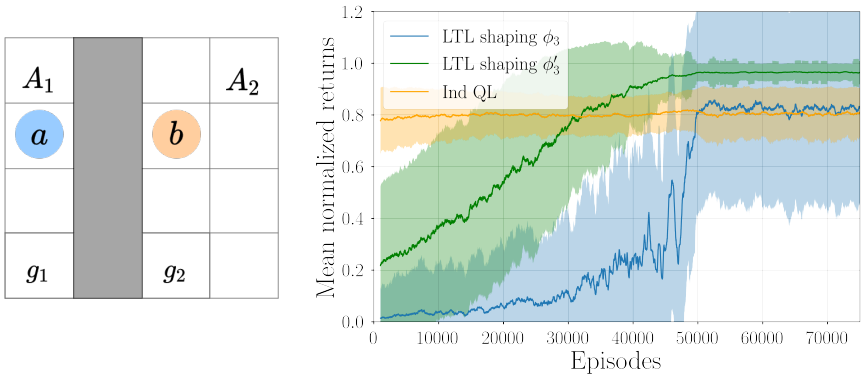
Impact on Learning Performance. This method is agnostic to the choice of a MARL algorithm because the reward shaping interacts with the learner only via inputs and outputs, and does not rely on the inner-workings of the learning algorithm $\mathbb{A}$. The convergence of this method may depend on the underlying algorithm. However, there is a lack of theoretical convergence guarantees for MARL algorithms in general. Thus, we focus on showing empirical convergence in our experiments in Section 5.4.

Because of the flexibility in the expression of an LTL specification, slightly different ways of specifying the same general tasks may lead to a different number of accepting transitions. The number of accepting transitions directly impacts the learning performance by providing more or less guidance with respect to progress in the LDBA. Note for example that the specifications for the motivating example (ϕ_3) and the rendez-vous benchmark here denoted by ϕ_3' (presented in section 5.4) have similar tasks which consist in having both agents synchronize at locations a and b then proceeds to the goal locations g_1 and g_2. However, the specifications differ with $\phi_3' = \mathtt{F}\ ((a \wedge b) \wedge \mathtt{X}\,\mathtt{F}\ ((g_1 \vee \mathtt{X}\,\mathtt{F}\ g_1) \wedge (g_2 \vee \mathtt{X}\,\mathtt{F}\ g_2)))$ resulting in the automaton shown in figure 5.9 in Section 5.4. Further analysis of both specifications shows that the winning region w_3 of ϕ_3' is covered by the winning region w_r for ϕ_3'; formally, $w_3 \subseteq w_r$. We show that in our experiments, a larger number of accepting transitions (i.e. more possible rewards during an accepting run) translates into better learning performance (fig. 5.5b).

5.4 Experiments

Our method is implemented in Python[1] and is built upon the single agent RL tool (CSRL) developed in [22] which uses the Owl library to synthesize the LDBA from the LTL specifications [96]. We also use the Spot library [97] to visualize the automata synthesized for each example in section 5.4. We applied our approach to three benchmarks including the previously discussed motivating example. The experiments were run on Windows

[1]The code is available at: https://github.com/IngyN/macsrl



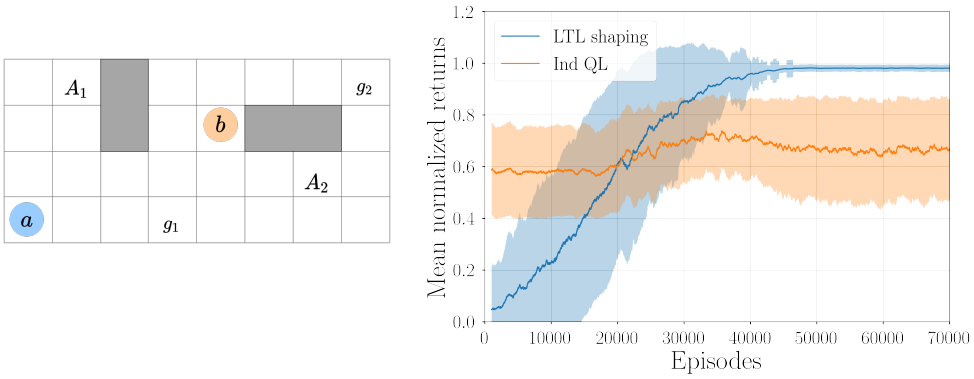
(A) Representation of the Environment

(B) Normalized Mean Return Learning Curve

FIGURE 5.5: Results for the Motivating Example

Subsystem for Linux (WSL2) running Ubuntu with an AMD Ryzen 7 CPU with 32 GB of RAM. All experiments terminated within one hour. We have not noticed a significant increase in time when running our method vs the same algorithm with no reward shaping. The main time complexity added by our method is the synthesis of the LDBAs and/or computation of the product MDP both of which only need to be computed once. For evaluation, we selected Independent Q-Learning [35] as a baseline but also as the underlying MARL algorithm for our method. Specifically, we used an ϵ-greedy policy with $\gamma = 0.999$ and $\gamma_B = 0.99$ (accepting transitions discount). Both the probability for exploration and the learning rate are gradually decreased over the training from 1.0 to (0.01 and 0.001) respectively. For presented benchmarks, the environments visualized in Figures 5.5a, 5.6a and 5.7a are non deterministic with a probability $p = 0.8$ of going to the desired location and a probability $(1 - p) = 0.2$ of ending up in another adjacent location. In all benchmarks, if an agent reaches a goal location (i.e. containing g in the label), they cannot move for the remainder of the episode. In order to compare the learning performance we normalized the returns such that the minimum reward received corresponds to 0 and the maximum corresponds to 1 for each method. Moreover, for all graphs, we average the normalized returns over the agents and apply smoothing using a rolling window of 1000 episodes.

Motivating Example. Our first benchmark is the motivating example described previously in Section 5.2. For the Independent Q-Learning baseline, we setup the independent reward function as follows: receive a reward of 2 if both agent are at a and b at the same time, 10 if they reach their goal. Notice that baseline reward function doesn't directly encourage agents pressing the buttons before reaching the goal. In contrast with our approach where the agents do not receive a reward for reaching the goal locations until both buttons have been pressed simultaneously. The results observed (fig.5.5b) show that for

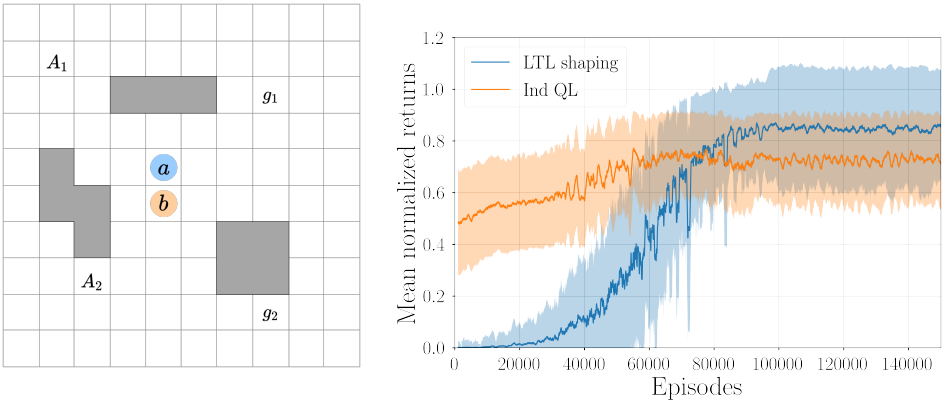


(A) Representation of the Environment (B) Normalized Mean Return Learning Curve

FIGURE 5.6: Results for the Flag Collection Example

both LTL specifications our method learns more reliably than the baseline with no reward shaping. In Section 5.3, we discuss two possible specifications featuring different automata structures. The LTL shaping curve in blue corresponds to our initial specification $\phi_3 = ((\texttt{F}\ a \wedge \neg g_1)\ \texttt{U}\ (a \wedge b)) \wedge ((\texttt{F}\ b \wedge \neg g_2)\ \texttt{U}\ (a \wedge b)) \wedge (\texttt{F}\ \texttt{G}\ g_1) \wedge (\texttt{F}\ \texttt{G}\ g_2)$. The LTL shaping curve in green corresponds to the specification $\phi_3' = \texttt{F}\ ((a \wedge b) \wedge \texttt{X}\,\texttt{F}\ ((g_1 \vee \texttt{X}\,\texttt{F}\ g_1) \wedge (g_2 \vee \texttt{X}\,\texttt{F}\ g_2)))$. The standard deviation over the rolling window is depicted by the shaded area for ϕ_3'. We remark that the standard deviation is larger than the Independent Q-Learning baseline method. We hypothesize that this is because for the LDBA generated from specification ϕ there only exists one accepting transition highlighted by a blue dashed line in Figure 5.4 compared to 5 accepting transitions as seen in Figure 5.9 for specification ϕ_3'. This results in the agents either receiving a normalized reward of 1 or 0 in an episode which explains the larger standard deviation. The learning curve for the reward shaping based on ϕ_3' shows both better normalized returns per episode and better convergence. This difference in learning performance demonstrates the importance of choosing an adequate formats for LTL tasks.

Flag collection. Our second benchmark is a flag collection scenario inspired by [33]. Previous work has demonstrated this kind of scenario to be challenging for RL agents even in the single agent case [28]. In this scenario (fig.5.6a), the agents must collect flags a and b then proceed to a goal location (g_1, g_2). Similarly to the previous example, in the baseline case agents receive a reward of 2 for collecting a flag and a reward of 10 for reaching a goal. In this benchmark, no instruction or constraints are given as to which agent should go to which goal or collect which flags. The LTL specification used for this example is the following: $\phi = \texttt{F}\ (a \wedge \texttt{F}\ (b \wedge (\texttt{F}\ (g_1 \vee \texttt{X}\,\texttt{F}\ g_1) \wedge \texttt{F}\ (g_2 \vee \texttt{X}\,\texttt{F}\ g_2))))$. Using the ($label \vee \texttt{X}\,\texttt{F}\ label$) format helps create a larger number of accepting transitions (6 accepting transitions, with an automaton with 7 states) guiding the reward shaping progress better (Figure 5.10 in section 5.4). In Figure 5.6b, our method performs much better than



(A) Representation of the Environment

(B) Normalized Mean Return Learning Curve

FIGURE 5.7: Rendez-vous

the baseline which fails to converge with the same number of training episodes. In our testing, increasing the number of training episodes has not yielded better convergence for the Independent Q-Learning baseline. Notice that unlike the previous benchmark, our method has a much smaller standard deviation after convergence.

Rendez-vous. Our third benchmark is a rendez-vous task where agents must meet in the adjacent locations a and b then proceed to goal locations g_1 and g_2 (fig. 5.7a). Similarly to the first benchmark, baseline agents receive a reward of 2 if both agents are at locations a and b at the same time step and 10 when each one reaches a goal location. In this scenario, agents are not assigned to a specific goal or meeting location, it is up to the learning algorithm to learn the optimal assignment. The LTL specification used for this benchmark is: $\phi = \texttt{F}\ ((a \wedge b) \wedge \texttt{X F}\ ((g_1 \vee \texttt{X F}\ g_1) \wedge (g_2 \vee \texttt{X F}\ g_2)))$. The synthesized LDBA is composed of 7 states including the trap state and contains 5 accepting transitions (fig. 5.9). Figure 5.7b shows that in this slightly larger benchmark both methods converge but both still have a fairly large standard deviation with the average return for the LTL shaping method being higher than the baseline method with shaping.

Experiment Automata. We now present the automata synthesized for each benchmark in our experiments (Sec. 5.4). The transitions annotated with a blue 0 are the accepting transitions. The automata seen here do not show the trap state to which agents would transition in the event of traces violating the specification. For example, in state 0 for the automaton in Figure 5.8, if either agent went to a goal location (g_1 or g_2) that would violate the specification and the common automaton would transition to the trap state.

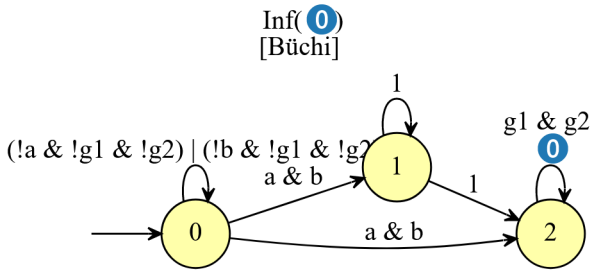


FIGURE 5.8: Automaton for Motivating Example with LTL specification ϕ_3

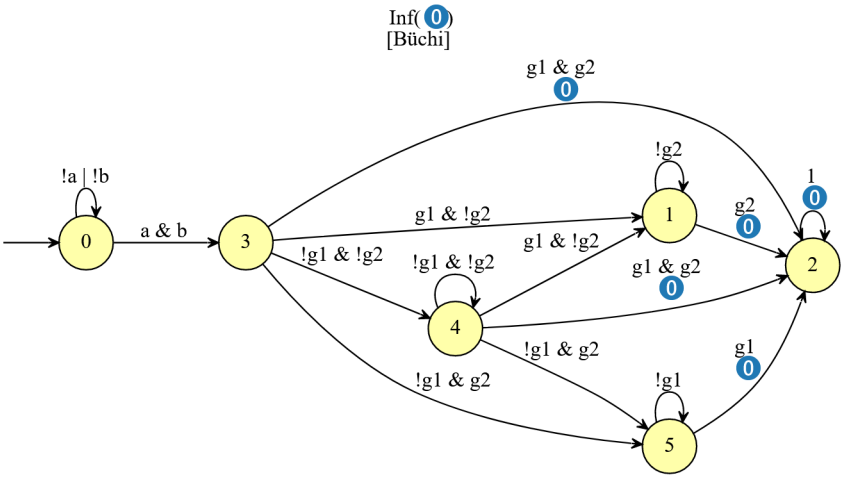


FIGURE 5.9: Automaton for the Rendez-vous Scenario and for the Motivating example with ϕ_3'

5.5 Discussion

In order to assign tasks to specific agents, we initially attempted to use separate automatons that would communicate the perceived labels. In the motivating example (fig. 5.2), the first agent's automaton would check for button b is triggered to be able to pass towards goal g_1. The second agent's automaton would include button a that can only be reached by the first agent. However, without explicitly adding the labels from the other agents in the automaton, the product MDP for any one agent is incomplete and cannot reach the accepting states. In our motivating example from fig. 5.2, this would also happen if agent A_2's LTL task was: $\phi_2 = (\neg g_2 \ \mathtt{U} \ a) \wedge \mathtt{F} \ \mathtt{G} \ g_2$. Then the second agent could wait until label a is triggered and immediately proceed to the goal in which case agent A_1 can never reach its accepting state.

In the case of Independent Q-Learning which is used as the underlying algorithm for our method, we argue that the augmented state $\langle s_t^i, q_t^i \rangle$ allows for indirect coordination

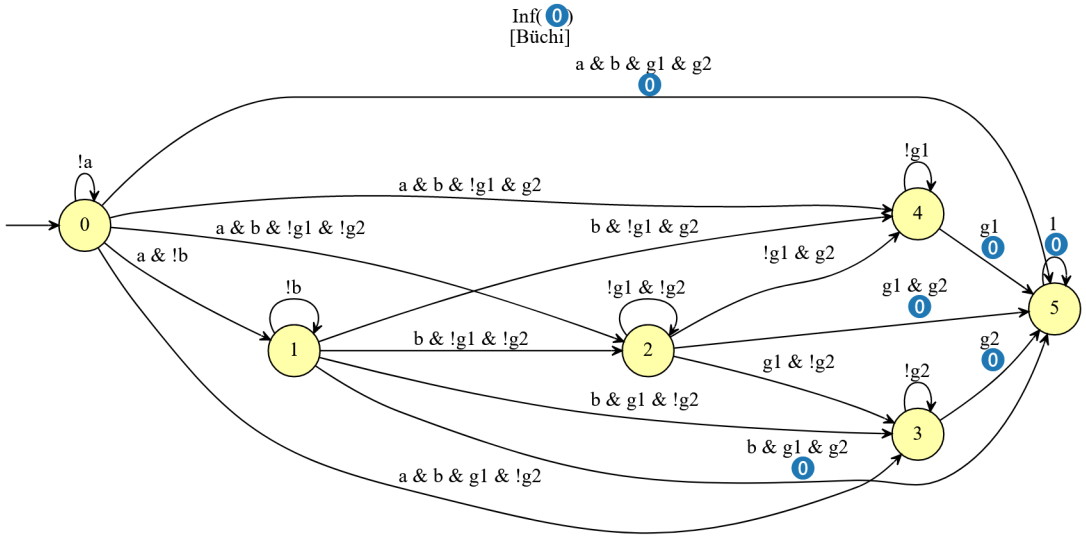


FIGURE 5.10: Automaton for the Flag Collection Scenario

by abstracting task progression more effectively. We can argue that if the joint LTL specification includes some task A and some task B, when task A is completed by an agent i then the automaton state changes, implicitly communicating to the other agents that task A is complete.

In this method, we assume that if agents choose conflicting ϵ-action then the first agent's choice in the agent ordering decides which one is being taken thereby enforcing a priority scheme based on agent ordering. For example, if we are using the LDBA from figure 5.1b one agent may choose to take the ϵ-action leading to q_1 and the other the ϵ-action leading to q_2, which transition should be taken? The more agents, we consider, the more likely conflicting ϵ-actions will be chosen. This is currently one of our main limitations and exploring how the use of a more dynamic agent priority scheme could be helpful to improve this work.

The difference in performance in our experiments shows that to have optimal reward shaping we may need to formulate the specification in such a way to increase the number of accepting transitions in the LDBA. Moreover, different methods for reward shaping create the reward function based on the automaton very differently. For example, in the approach presented in [28], the authors use heuristics and attempt to estimate the progress within the LDBA to better guide the learning process.

For the experiments section, we also attempted to demonstrate that this method could work with another algorithm (we tried with shared-experience Q-learning [98]) in a slightly modified foraging gym environment [99]. The agents were unable to learn in our attempts. However, we believe it was an issue with the implementation of the necessary modifications rather than a failure of our method.

5.6 Summary

In this chapter, we present an approach to multi-agent reward shaping that can successfully learn LTL defined tasks. The semi-centralized reward shaping approach addresses the issue of scalability and the need for coordination between agents. We also demonstrate how the use of LTL in this approach can allow for flexible task specification and assignment. In our experimental results, we highlight that our method can achieve encouraging learning performance when compared to Independent Q-Learning alone.

In the future, we may look into ways to improve the logic-based reward shaping mechanism further by exploring how this method would perform in the context of lossy or restricted communication. It is our belief that a slightly modified main approach would be a useful tool in this context. Another idea would be to use a hierarchical approach for team based tasks. Furthermore, we may seek to revisit the decentralized approach and analyze how it can be achieved in a modified problem. Finally, we may explore the use of our method with a better way of determining progress through the automaton rather than waiting until agents reach accepting transitions. A denser reward function as presented in [32] has the potential to improve learning performance and could be adapted to our method.

While MARL excels at supporting a large number agents and expansive continuous environments, this comes at the cost of precision since many MARL algorithms rely on functional approximation to be scalable. Alternatively, probabilistic model checking offers logic-based sequential decision making tools resulting in interpretable policies when provided with an MDP model of the environment. In the following chapters, we explore informative safe decision-making in probabilistic model checking.

Chapter 6

Distributional Probabilistic Model Checking

In the two previous chapters, we have explored scalable methods to achieve both soft and hard safety constraints in multi-agent reinforcement learning (MARL). Nevertheless, MARL methods operate as a black-box making it hard to rigorously evaluate the learned policy. Furthermore, a complete safety guarantee may be overly conservative or unrealistic when systems (e.g., robotics) exhibit probabilistic and non-deterministic behavior due to inherent uncertainty (e.g., sensor noise, human interactions, weather).

Probabilistic model checking offers tools to both model and verify this type of challenging environment. It also supports not only their verification against specifications in temporal logic, but also synthesis of optimal controllers (policies). Commonly used models include discrete-time Markov chains (DTMCs) and Markov decision processes (MDPs). A range of verification techniques for these, and other models, are supported by widely used probabilistic model checkers such as PRISM [9] and Storm [10]. In this chapter, we aim to provide safe sequential decision making methods by evaluating the cost distribution of any given policy and synthesizing risk-sensitive policies.

Risk-aware distributional measures such as *conditional value-at-risk* (CVaR) [47] address this by minimizing the costs that occur above a specified point in the tail of distribution. Within probabilistic model checking, the use of *quantiles* has been proposed [39]–[42] to reason about cost or reward distributions.

In this chapter, we develop a *distributional probabilistic model checking* approach, which computes and reasons about the full distribution over the reward associated with a DTMC or MDP. More precisely, we consider the reward accumulated until a specification in co-safe LTL is satisfied, the latter providing an expressive means to specify, for example, a multi-step task to be executed by a robot [100], or a sequence of events leading to a system failure. We propose a temporal logic based specification for such distributional queries.

For a DTMC, we perform model checking of these queries by generating a precise representation of the distribution, up to an arbitrary, pre-specified level of accuracy (the distribution is discrete, but with possibly countable infinite support, so at least some level of truncation is typically required). This is based on a graph analysis followed by a forward numerical computation. From this, we can precisely compute a wide range of useful properties, such as the mean, variance, mode or various risk-based measures.

For an MDP, we instead aim to optimize such properties over all policies. In this chapter, we focus on optimizing the expected value or CVaR, whilst generating the full reward distribution for each state of the MDP. This is done using *distributional value iteration* (DVI) [56], which can be seen as a generalization of classical value iteration. Rather than computing a single scalar value (e.g., representing the optimal expected reward) for each MDP state, DVI associates a full distribution with each state, replacing the standard Bellman equation with a distributional Bellman equation.

We consider two types of DVI algorithms, namely *risk-neutral* DVI for optimizing the expected value and *risk-sensitive* DVI for optimizing CVaR. Risk-neutral DVI can be shown to converge to a deterministic, memoryless optimal policy, if a unique one exists [56]. For CVaR, memoryless policies do not suffice for optimality, but risk-sensitive DVI does converge for a product MDP that incorporates a (continuous) slack variable representing a reward budget [58]. To improve computational efficiency, we present a risk-sensitive DVI algorithm based on a discretization of the slack variable, and show that the algorithm converges to a CVaR optimal policy for increasingly precise discretizations.

For both DVI algorithms, in practice it is necessary to use approximate distributional representations. We consider the use of categorical and quantile representations. This can impact both optimality and the precision of computed distributions but, for the latter, we can construct the DTMC induced by generated MDP policies and use our precise approach to generate the correct distribution. Finally, we implement our distributional probabilistic model checking framework as an extension of the PRISM model checker [9] and explore the feasibility and performance of the techniques on a range of benchmarks.

6.1 Distributional Probabilistic Model Checking

We formulate our approach as a *distributional* extension of probabilistic model checking, which is a widely used framework for formally specifying and verifying quantitative properties of probabilistic models. In particular, we build upon existing temporal logics in common use. The core property we consider is the probability distribution over the amount of reward (or cost) that has been accumulated until some specified sequence of events occurs (which could constitute, for example, the successful completion of a task by a robot, or a combination of events that leads to a system failure). To represent events,

we use the co-safe fragment of linear temporal logic (LTL) as described in Definition 6. The key ingredient of our temporal logic specifications is a *distributional query*, which gives a property (such as the expected value, or variance) of the distribution over the accumulated reward until an event's occurrence.

Definition 7 (Distributional query)**.** For a DTMC, a *distributional query* takes the form $\mathtt{R}^{\mathsf{f}(r)}_{=?}[\,\psi\,]$, where r is a reward structure, f is a random variable property (e.g., $\mathbb{E}, \mathrm{Var}, \mathrm{s.d.}, \mathrm{mode}, \mathsf{VaR}, \mathsf{CVaR}$), and ψ is a formula in co-safe LTL.

Examples of properties that can be expressed in this framework include:

- $\mathtt{R}^{\mathbb{E}(r_{time})}_{=?}[\,\mathtt{F}\ \mathit{end}\,]$ - the expected time until an algorithm terminates;
- $\mathtt{R}^{\mathrm{Var}(r_{energy})}_{=?}[\,\mathtt{F}\ (\mathit{goal}_1 \wedge \mathtt{F}\ \mathit{goal}_2)\,]$ - the variance in energy consumption until a robot visits location goal_1 followed by location goal_2.
- $\mathtt{R}^{\mathrm{mode}(r_{coll})}_{=?}[\,\mathtt{F}\ \mathit{sent}_1 \vee \mathtt{F}\ \mathit{sent}_2\,]$ - the most likely number of packet collisions before a communication protocol successfully sends one of two messages.

For an MDP, the goal is to optimize some random variable property f over the policies of the MDP. In this chapter, we restrict our attention to two particular cases, expected value ($\mathbb{E}$) and conditional value-at-risk (CVaR), and call these *distributional optimization queries*.

Definition 8 (Distributional optimization query)**.** For an MDP, a *distributional optimization query* takes the form $\mathtt{R}^{\mathsf{f}(r)}_{\mathrm{opt}=?}[\,\psi\,]$, where r is a reward structure, $\mathsf{f} \in \{\mathbb{E}, \mathsf{CVaR}\}$, $\mathrm{opt} \in \{\min, \max\}$ and ψ is a formula in co-safe LTL. For the resulting policy, we can perform *policy evaluation* on the induced DTMC using one or more other distributional queries $\mathtt{R}^{\mathsf{f}'(r')}_{=?}[\,\psi'\,]$.

An example optimization query is $\mathtt{R}^{\mathsf{CVaR}_{0.9}(r_{time})}_{\min=?}[\,\mathtt{F}\ \mathit{goal}\,]$, which minimizes the conditional value-at-risk with respect to the time for a robot to reach its goal.

Semantics. A distributional query $\mathtt{R}^{\mathsf{f}(r)}_{=?}[\,\psi\,]$ is evaluated on a DTMC $\mathcal{D}$, and a distributional optimization query $\mathtt{R}^{\mathsf{f}(r)}_{\mathrm{opt}=?}[\,\psi\,]$ on an MDP $\mathcal{M}$, in each case via a random variable for the reward accumulated from its initial state:

$$\begin{aligned}
\mathtt{R}^{\mathsf{f}(r)}_{=?}[\,\psi\,] &= \mathsf{f}(X^{r,\psi}_{\mathcal{D}}) \\
\mathtt{R}^{\mathsf{f}(r)}_{\min=?}[\,\psi\,] &= \inf_{\pi \in \Sigma_{\mathcal{M}}} \mathsf{f}(X^{r,\psi}_{\mathcal{M},\pi}) \\
\mathtt{R}^{\mathsf{f}(r)}_{\max=?}[\,\psi\,] &= \sup_{\pi \in \Sigma_{\mathcal{M}}} \mathsf{f}(X^{r,\psi}_{\mathcal{M},\pi})
\end{aligned}$$

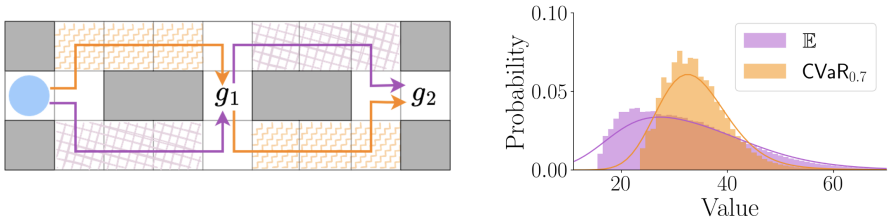


FIGURE 6.1: The "mud & nails" example. Left: Map of the terrain to navigate, with two policies that minimize expected cost and conditional value-at-risk to visit g_1 and then g_2. Right: The corresponding distributions over cost.

where the random variables $X^{r,\psi}_{\mathcal{D}} : \mathit{IPaths}_{\mathcal{D}} \to \mathbb{R}$ and $X^{r,\psi}_{\mathcal{M},\pi} : \mathit{IPaths}_{\mathcal{M}} \to \mathbb{R}$ are:

$$X^{r,\psi}_{\mathcal{D}}(\omega) \;=\; X^{r,\psi}_{\mathcal{M},\pi}(\omega) \;=\; \begin{cases} r(\omega, k_\psi - 1) & \text{if } \omega \models \psi \\ \infty & \text{otherwise} \end{cases}$$

and $k_\psi = \min\{k \mid (\omega, k) \models \psi\}$ is the length of the shortest good prefix for ψ.

Example 3. We illustrate our framework with an example of an autonomous robot navigating within a risky environment ("mud & nails"), modelled as an MDP. Figure 6.1 illustrates the scenario: the robot starts in the leftmost location (blue circle), and may pass through two types of terrain, mud (orange zigzag) and ground littered with nails (purple hatching). Obstacles are drawn as grey blocks. The cost of navigation is, by default, 1 per step, obstacles incurring an additional cost of 35. In the "nails" terrain, there is a probability of 0.2 of hitting a nail, which then incurs a cost of 5; navigating the "mud" terrain is safer but slower: it incurs a cost of 3 per step. Consider the total cost to visit g_1 and then g_2. Given a reward structure *cost* encoding individual costs as above, we can aim to minimize either the expected cost or the conditional value-at-risk, using queries $\mathtt{R}^{\mathbb{E}(cost)}_{\min=?}[\,\mathtt{F}\ (g_1 \wedge \mathtt{F}\ g_2)\,]$ or $\mathtt{R}^{\mathsf{CVaR}_{0.7}(cost)}_{\min=?}[\,\mathtt{F}\ (g_1 \wedge \mathtt{F}\ g_2)\,]$. Figure 6.1 also shows the resulting policies, plotted on the map in purple and orange, respectively, and the corresponding probability distributions over cost. We can analyze each policy with further distributional queries, e.g., $\mathtt{R}^{\mathtt{f}(cost)}_{=?}[\,\mathtt{F}\ g_1\,]$ for $\mathtt{f} = \{\mathbb{E}, \mathrm{Var}\}$ to evaluate the mean and variance of the cost to reach g_1. ■

6.2 Distributional Model Checking Algorithms

We now describe algorithms for distributional probabilistic model checking, i.e., to evaluate distributional queries of the form $\mathtt{R}^{\mathtt{f}(r)}_{=?}[\,\psi\,]$ for a DTMC or $\mathtt{R}^{\mathtt{f}(r)}_{\mathrm{opt}=?}[\,\psi\,]$ for an MDP. Following the semantics given in Section 6.1, for a DTMC $\mathcal{D}$, this necessitates generating the probability distribution of the random variable $X^{r,\psi}_{\mathcal{D}}$, corresponding to reward structure r and LTL formula ψ, on $\mathcal{D}$. The value $\mathtt{f}(X^{r,\psi}_{\mathcal{D}})$ can then be evaluated on the

distribution for any f. For an MDP $\mathcal{M}$, we aim to find a policy π^* which optimizes the value $\mathsf{f}(X^{r,\psi}_{\mathcal{M},\pi})$ over policies π.

For both classes of model, in standard fashion, we reduce the problem to the simpler case where ψ is a *reachability* formula by constructing an automaton product. More precisely, we build a deterministic finite automaton (DFA) $\mathcal{A}_\psi$ representing the "good" prefixes of co-safe LTL formula ψ, and then construct a DTMC-DFA product $\mathcal{D} \otimes \mathcal{A}_\psi$ or MDP-DFA product $\mathcal{M} \otimes \mathcal{A}_\psi$ with state space $S \times Q$, where S is the state space of the original model and Q the states of the DFA. There is a one-to-one correspondence between paths (and, for MDPs, policies) in the original model and the product model [101].

Hence, in what follows, we restrict our attention to computing the probability distributions for random variables defined as the reward to reach a target set of states $T \subseteq S$, describing first the case for a DTMC and then the cases for risk-neutral ($\mathsf{f} = \mathbb{E}$) and risk-sensitive ($\mathsf{f} = \mathsf{CVaR}$) optimization for an MDP. For the latter two, for presentational simplicity, we focus on the case of minimization, but it is straightforward to adapt the algorithms to the maximizing case.

6.2.1 Forward Distribution Generation for DTMCs

We fix a DTMC $\mathcal{D}$, reward structure r and set of target states T. In this section, we describe how to compute the probability distribution for the reward r accumulated in $\mathcal{D}$ until T is reached, i.e., for the random variable $X^{r,\mathbf{F}\,T}_{\mathcal{D}}$. We denote this distribution by μ. Note that, since individual rewards are integer-valued, and are summed along paths, μ is a discrete distribution.

We compute the distribution in a forward manner, up to a pre-specified accuracy ε, using Algorithm 4. First, note that the reward accumulated along a path that never reaches the target T is defined to be ∞ (see Section 6.1). Probabilistic model checking algorithms typically compute the *expected* reward to reach a target T from a state s, which is therefore infinite if s has a non-zero probability of not reaching T. Here, we have to take slightly more care since there may be states from which there is a non-zero probability of both accumulating finite and infinite reward. This means that μ is a distribution over $\mathbb{N}_\infty$.

Algorithm 4 first identifies the states S_∞ of $\mathcal{D}$ from which the probability of accumulating infinite reward is 1, which are those in bottom strongly connected components (BSCCs) of $\mathcal{D}$ that do not intersect with T. It then computes a discrete distribution $\mu_\times$ over $S \times \mathbb{N}_\infty$ where, at the kth iteration, $\mu_\times(s,i)$ is the probability of being in state s and having accumulated reward i after k steps. A new version $\mu'_\times$ is computed at each step. Abusing notation, we write distributions as lists $\{x_1 \mapsto p_1, \dots\}$ of the elements x_j of their support and their probabilities p_j. We also keep track of the probabilities $p_{\overline{T}}$ and p_∞ of, by the kth iteration, *not* having reached the target set T and being in S_∞, respectively.

Algorithm 4: Forward distribution generation for DTMCs

Input : DTMC $\mathcal{D} = (S, s_0, P, AP, L)$, reward structure r, target set $T \subseteq S$, accuracy $\varepsilon \in \mathbb{R}_{>0}$

Output: The discrete probability distribution μ for $X_{\mathcal{D}}^{r,\mathbf{F}\,T}$.

$S_\infty \leftarrow \{s \in S \mid s \text{ is in a BSCC } C \subseteq S \text{ with } C \cap T = \emptyset\}$
$\mu_\times \leftarrow \delta_{(s_0,0)}$
$p_{\overline{T}} = 1$; $p_\infty = 0$
while $p_{\overline{T}} - p_\infty > \varepsilon$ **do**
  $\mu'_\times \leftarrow \{\}$
  $p_{\overline{T}} \leftarrow 0$
  for $((s,i) \mapsto p_{s,i}) \in \mu_\times$ **do**
    if $s \in T$ **then**
      $\mu'_\times(s,i) \leftarrow \mu'_\times(s,i) + p_{s,i}$
    else
      for $(s' \mapsto p_{s'}) \in P(s,\cdot)$ **do**
        if $s' \notin T$ **then**
          $p_{\overline{T}} \leftarrow p_{\overline{T}} + p_{s,i} \cdot p_{s'}$
        if $s' \notin S_\infty$ **then**
          $\mu'_\times(s', i + r(s)) \leftarrow \mu'_\times(s', i + r(s)) + p_{s,i} \cdot p_{s'}$
        else
          $p_\infty \leftarrow p_\infty + \mu_\times(s,i) \cdot p_{s'}$
  $\mu_\times \leftarrow \mu'_\times$
return $\{i \mapsto p_i \,|\, p_i = \sum_s \mu_\times(s,i)\} \cup \{\infty \mapsto p_\infty\}$

The distribution μ is finally computed by summing $\mu_\times(s,i)$ values over all states and can be analyzed with additional distributional properties.

Correctness and convergence. Let μ be the exact distribution for $X_{\mathcal{D}}^{r,\mathbf{F}\,T}$ and $\hat{\mu}$ be the one returned by Algorithm 4, using accuracy $\varepsilon > 0$. We have:

$$\mu(i) \leq \hat{\mu}(i) \leq \mu(i) + \varepsilon \qquad \text{for all } i \in \mathbb{N}_\infty \tag{6.1}$$

Note that the support of μ may be (countably) infinite, but $\hat{\mu}$ is finite by construction. In this case, the total truncation error is also bounded by ε: if $\hat{k} \in \mathbb{N}$ is the maximum finite value in the support of $\hat{\mu}$, then $\sum_{\hat{k}<i<\infty} \mu(i) \leq \varepsilon$.

To see the correctness of Equation (6.1), observe that $\hat{\mu}(i)$ is ultimately computed from the sum of the values $\sum_s \mu_\times(s,i)$ in Algorithm 4, the total value of which is non-decreasing since rewards are non-negative. In any iteration, at most $p_{\overline{T}} - p_\infty$ will be added to any value $\mu_\times(s,i)$ and, on termination, $p_{\overline{T}} - p_\infty \leq \varepsilon$. Convergence is guaranteed for any $\varepsilon > 0$: since we separate the states S_∞ in non-target BSCCs, within k iterations, the combined probability of having reached T (i.e., $1 - p_{\overline{T}}$) or reaching S_∞ (i.e., p_∞) tends to 1 as $k \to \infty$.

Algorithm 5: Risk-Neutral Distributional Value Iteration

Input : MDP $\mathcal{M} = (S, s_0, A, P, AP, L,)$, reward structure r, target set $T \subseteq S$, and convergence threshold $\epsilon \in \mathbb{R}_{>0}$

Output: optimal policy π^* for query $\mathtt{R}^{\mathbb{E}(r)}_{\min=?}[\mathtt{F}\, T\,]$, distribution μ_{s_0} under π^*

1 **foreach** $s \in S$ **do**
2 $\mu_s \leftarrow \delta_0$
3 **while** $e > \epsilon$ **do**
4 **foreach** $s \in S \setminus T$ **do**
5 **foreach** $a \in A(s)$ **do**
6 $\eta(s,a) :\overset{D}{=} \mathtt{proj}(r(s,a) + \sum_{s' \in S} P(s,a,s') \cdot \mu_{s'})$
7 $\pi^*(s) \leftarrow \arg\min_{a \in A(s)} \mathbb{E}\big(X | X \sim \eta(s,a)\big)$
8 $\mu'_s \leftarrow \eta(s, \pi^*(s))$
9 $e \leftarrow \sup_{s \in S \setminus T} d(\mu_s, \mu'_s)$
10 **foreach** $s \in S \setminus T$ **do**
11 $\mu_s \leftarrow \mu'_s$
12 **return** π^* and μ_{s_0}

6.2.2 Risk-Neutral Distributional Value Iteration for MDPs

In this section, we present a *risk-neutral* DVI method for computing value distributions of states of an MDP $\mathcal{M}$ under an optimal policy that minimizes the *expected* cumulative reward to reach a target set $T \subseteq S$, i.e., minimizes $\mathbb{E}(X^{r,\mathtt{F}\,T}_{\mathcal{M},\pi})$ for random variables $X^{r,\mathtt{F}\,T}_{\mathcal{M},\pi}$ of MDP policies π.

In contrast to the case for DTMCs, since we now consider expected values, we assume that there exists an optimal policy with finite expected reward, i.e., which reaches the target set T with probability 1. This can be checked efficiently with an analysis of the underlying graph of the MDP [102].

The risk-neutral DVI method is shown in Algorithm 5. For each MDP state $s \in S$, it initializes its value distribution μ_s to Dirac distribution δ_0. The algorithm loops through lines 3-11 to update value distributions of any non-target state $s \in S \setminus T$ as follows. For each available action $a \in A(s)$ in state s, a value distribution is obtained via the distributional Bellman equation shown in line 6 then projected to $\eta(s,a)$ to match the chosen representation (see Def. 1, 2). The optimal action $\pi^*(s)$ in state s is the one that achieves the minimal expected value of $\eta(s,a)$. The updated value distribution μ'_s of state s is given by $\eta(s, \pi^*(s))$. The algorithm terminates when the supremum of distributional distance $d(\mu_s, \mu'_s)$ across all states (the choice of metrics is discussed later) is less than the convergence threshold ϵ. Unlike the accuracy ε for Algorithm 4, this threshold ϵ does *not* provide a guarantee on the precision of the result after convergence (similar issues occur in classical value iteration for MDPs [103]).

Example 4. We can check $\mathtt{R}^{\mathbb{E}(cost)}_{\min=?}[\,\mathtt{F}\ (g_1 \wedge \mathtt{F}\ g_2)\,]$ on our running example using risk-neutral DVI on a product MDP. The resulting optimal policy and distribution are shown in purple in Figure 6.1. ■

Distributional approximation. To enable a practical implementation of the algorithm, we need a probability distribution representation with finitely many parameters to store value distributions in memory. Here, we can adopt the categorical (see Definition 1) or quantile (see Definition 2) representations. Specifically, we need to apply the categorical or quantile projection (see [56]) after each update of the distributional Bellman equation (line 6). We use the supremum Cramér distance $\overline{\ell}_2$ for categorical representations and the supremum Wasserstein distance $\overline{w}_1$ for quantile representations as the distance metric in line 9 (see [56] for distributional distance definitions).

Policy convergence. When there exists a unique risk-neutral optimal policy, Algorithm 5 is guaranteed to converge to it (following [56, Theorem 7.9]). However, when there are multiple optimal policies, risk-neutral DVI may fail to converge (see [56, Section 7.5]). Furthermore, inaccuracies due the use of distributional approximations could potentially lead to a sub-optimal policy being chosen. To mitigate this, for either categorical or quantile representations, increasing the number m of atoms used yields tighter approximation error bounds [56].

6.2.3 Risk-Sensitive Distributional Value Iteration for MDPs

By contrast to risk-neutral policies that seek to minimize the expected reward, *risk-sensitive* policies make decisions accounting for risk properties. In this section, we present a risk-sensitive DVI method for minimizing the *conditional value-at-risk* of reaching a target set in an MDP $\mathcal{M}$, i.e., minimizing $\mathsf{CVaR}_\alpha(X^{r,\mathtt{F}\,T}_{\mathcal{M},\pi})$ for random variables $X^{r,\mathtt{F}\,T}_{\mathcal{M},\pi}$ of MDP policies π. Our method follows a key insight from [58], [104] that conditional value-at-risk can be represented as the solution of a convex optimization problem.

Lemma 1 (Dual Representation of CVaR [58], [104])**.** *Let $[x]^+$ denote the function that is 0 if $x < 0$, and x otherwise. Given a random variable X over the probability space $(\Omega, \mathcal{F}, \Pr)$, it holds that:*

$$\mathsf{CVaR}_\alpha(X) = \min_{b\in\mathbb{R}} \left\{ b + \frac{1}{1-\alpha}\mathbb{E}\big(\,[X-b]^+\,\big) \right\}, \tag{6.2}$$

and the minimum-point is given by $b^* = \mathsf{VaR}_\alpha(X)$. □

Intuitively, the *slack variable* $b \in [V_{\min}, V_{\max}]$ encodes the risk budget and possible $\mathsf{VaR}_\alpha(X)$ values. Since $\mathsf{VaR}_\alpha(X) \in [V_{\min}, V_{\max}]$, the slack variable is similarly bounded by the minimum and maximum possible accumulated reward within the MDP, respectively. We assume that the reward values are bounded and the probability of reaching

Algorithm 6: Risk-Sensitive Distributional Value Iteration

Input : MDP $\mathcal{M} = (S, s_0, A, P, AP, L,)$, reward structure r, target set $T \subseteq S$, risk level α, slack variable set B, convergence threshold $\epsilon \in \mathbb{R}_{>0}$

Output: optimal policy π^* for query $\mathtt{R}^{\mathsf{CVaR}_\alpha(r)}_{\min=?}[\mathtt{F}\, T]$, distribution μ_{s_0} under π^*

1 Construct product MDP $\mathcal{M}^{\mathsf{b}} = (S \times B, \{s_0\} \times B, A, P^{\mathsf{b}}, AP, L^{\mathsf{b}})$
2 $\mu_{\langle s,b\rangle} \leftarrow \delta_0$, $\forall \langle s, b\rangle \in S \times B$
3 **while** $e > \epsilon$ **do**
4 **foreach** $\langle s, b\rangle \in (S \setminus T) \times B$ **do**
5 **foreach** $a \in A(s)$ **do**
6 $\eta(\langle s, b\rangle, a) :\overset{D}{=} \mathtt{proj}(r(s, a) + \sum_{\langle s',b'\rangle \in S\times B} P^{\mathsf{b}}(\langle s, b\rangle, a, \langle s', b'\rangle) \cdot \mu_{\langle s',b'\rangle})$
7 $\pi^{\mathsf{b}}(\langle s, b\rangle) \leftarrow \arg\min_{a\in A(s)} \mathbb{E}\big([X - b]^+ \,|X \sim \eta(\langle s, b\rangle, a)\big)$
8 $\mu'_{\langle s,b\rangle} \leftarrow \eta(\langle s, b\rangle, \pi^{\mathsf{b}}(\langle s, b\rangle))$
9 $e \leftarrow \sup_{\langle s,b\rangle\in(S\setminus T)\times B} d(\mu_{\langle s,b\rangle}, \mu'_{\langle s,b\rangle})$
10 $\mu_{\langle s,b\rangle} \leftarrow \mu'_{\langle s,b\rangle}$, $\forall \langle s, b\rangle \in (S \setminus T) \times B$
11 $\bar{b}^* \leftarrow \arg\min_{\bar{b}\in B} \mathsf{CVaR}_\alpha(X|X \sim \mu_{\langle s_0,\bar{b}\rangle}), \forall \bar{b} \in B$
12 $\pi^* \leftarrow$ policy π^{b} of the product MDP $\mathcal{M}^{\mathsf{b}}$ with initial state fixed to $\langle s_0, \bar{b}^*\rangle$
13 **return** π^* and $\mu_{\langle s_0,\bar{b}^*\rangle}$

the target states is 1, therefore $V_{\min}$ and $V_{\max}$ are also bounded. To enable efficient computation, we consider a discrete number of values for b. More precisely, we define a set B with n evenly-spaced atoms $b_1 < \cdots < b_n$ such that $b_1 = V_{\min}$, $b_n = V_{\max}$, and the stride between two successive atoms is $\varsigma_n = \frac{V_{\max} - V_{\min}}{n-1}$. Based on Lemma 1, determining the optimal slack variable value b^* requires computation of VaR_α for the distribution, which cannot be obtained *a priori*. Thus, we consider all possible risk budgets.

Algorithm 6 illustrates the proposed method. We construct a product MDP model $\mathcal{M}^{\mathsf{b}} = (S \times B, \{s_0\} \times B, A, P^{\mathsf{b}}, AP, L^{\mathsf{b}})$. Unlike the product MDP defined in Section 6.1, this MDP has multiple initial states, one state $\langle s_0, \bar{b}\rangle$ for each risk budget $\bar{b} \in B$, where s_0 is the initial state of the MDP $\mathcal{M}$. For each transition $s \xrightarrow{a} s'$ in $\mathcal{M}$ with $P(s, a, s') > 0$, there is a corresponding transition $\langle s, b\rangle \xrightarrow{a} \langle s', b'\rangle$ in $\mathcal{M}^{\mathsf{b}}$, where b' is obtained by rounding down the value of $b - r(s, a)$ to the nearest smaller atom in B and $P^{\mathsf{b}}(\langle s, b\rangle, a, \langle s', b'\rangle) = P(s, a, s')$. The labelling function is given by $L^{\mathsf{b}}(\langle s, b\rangle) = L(s)$. Next, in lines 2-10, Algorithm 6 initializes and updates the value distribution of each augmented state $\langle s, b\rangle \in S \times B$ in the product MDP $\mathcal{M}^{\mathsf{b}}$ in a similar fashion to the risk-neutral DVI described in Section 6.2.2. However, when choosing the optimal action (line 7), Algorithm 6 adopts a different criterion that minimizes $\mathbb{E}([X - b]^+)$ based on the dual representation of CVaR (see Equation 6.2).

Different choices of the initial risk budget $\bar{b}$ lead to various value distributions. Once DVI on the product MDP $\mathcal{M}^{\mathsf{b}}$ converges, the algorithm selects the optimal risk budget, denoted by $\bar{b}^*$, that yields the minimum CVaR of all possible initial value distributions $\mu_{\langle s_0,\bar{b}\rangle}$.

Finally, the algorithm returns the optimal policy π^* resulting from the risk-sensitive DVI on the product MDP $\mathcal{M}^{\mathsf{b}}$ with initial state $\langle s_0, \bar{b}^* \rangle$, and returns the distribution $\mu_{\langle s_0, \bar{b}^* \rangle}$.

Correctness and convergence. Let $X^{r,\mathbf{F}\,T}_{\mathcal{M},\pi}$ be the random variable for accumulation of reward r until reaching states T under policy π of an MDP $\mathcal{M}$.

We first restate a closely related theorem from [58].

Theorem 1. [58, Theorem 4.5, adapted] Assuming a continuous slack variable $b \in \mathbb{R}$, there exists a solution $\bar{b}^*$ when minimizing, over values of b, the inner formula $\mathbb{E}^{\pi}([X^{r,\mathbf{F}\,T}_{\mathcal{M},\pi} - b]^+)$ of Equation 6.2, for which the optimal policy for $\mathsf{CVaR}_\alpha(X^{r,\mathbf{F}\,T}_{\mathcal{M}^b,\pi})$ from initial state $\langle s_0, \bar{b}^* \rangle$ in the augmented MDP $\mathcal{M}^b$ solves the following optimization problem for a fixed $\alpha \in [0,1]$ in the original MDP $\mathcal{M}$:

$$\inf_{\pi \in \Pi} \mathsf{CVaR}_\alpha(X^{r,\mathbf{F}\,T}_{\mathcal{M},\pi})$$

Theorem 1 guarantees that risk-sensitive DVI using the inner formula for the augmented MDP $\mathcal{M}^b$ leads to the optimal policy. Moreover, the optimal policy for the augmented model $\mathcal{M}^b$ is also optimal for the original MDP $\mathcal{M}$. However, this theorem assumes a continuous slack variable. Algorithm 6, which uses a discretized slack variable (i.e., the set of atoms B is finite), converges to the same optimal policy π^{b} as $|B|$ increases. In the following, we prove that, as the number of atoms used for $|B|$ increases (i.e., stride ς_n decreases), the optimal policy for a finite set B becomes closer to the optimal policy with a continuous slack variable in terms of CVaR values.

Lemma 2. *Let π_1 denote the optimal policy for minimizing $\mathsf{CVaR}_\alpha(X^{r,\mathbf{F}\,T}_{\mathcal{M},\pi})$, which is obtained with a continuous slack variable. Let π_2 denote the optimal policy returned by Algorithm 6 where B is a finite set of n evenly-spaced atoms with stride ς_n. It holds that $\mathsf{CVaR}_\alpha(X^{r,\mathbf{F}\,T}_{\mathcal{M},\pi_2}) - \mathsf{CVaR}_\alpha(X^{r,\mathbf{F}\,T}_{\mathcal{M},\pi_1}) = \mathcal{O}(\varsigma_n)$. As ς_n tends to 0 (i.e., $|B|$ increases), π_2 converges to the CVaR optimal policy.* □

Proof. Recall that, when building the product MDP in Algorithm 6, we determine the slack variable value b' for a successor state by rounding down the value of $b - r(s,a)$ to the nearest smaller atom in B. More precisely,

$$b' = V_{\min} + \varsigma_n \cdot \left\lfloor \frac{\max(V_{\min}, b - r) - V_{\min}}{\varsigma_n} \right\rfloor$$

This is the main source of approximation errors introduced by the discretization of the slack variable. Following the dual representation of CVaR given in Lemma 1, each slack

variable value update would introduce the error of:

$$\begin{aligned} & \mathsf{CVaR}_\alpha(X^{r,\mathbf{F}\,T}_{\mathcal{M},\pi_2}) - \mathsf{CVaR}_\alpha(X^{r,\mathbf{F}\,T}_{\mathcal{M},\pi_1}) \\ = \;& \big(b' - \max(V_{\min}, b-r)\big) \\ + \;& \frac{1}{1-\alpha}\Big(\mathbb{E}(\big[X-b'\big]^+) - \mathbb{E}([X-\max(V_{\min}, b-r)]^+)\Big) \end{aligned}$$

Let $\beta := \max(V_{\min}, b-r)$. The first term yields,

$$\begin{aligned} b'-\beta &= V_{\min} + \varsigma_n \cdot \left\lfloor \frac{\beta - V_{\min}}{\varsigma_n} \right\rfloor - \beta \\ &= \varsigma_n \cdot \left(\left\lfloor \frac{\beta - V_{\min}}{\varsigma_n} \right\rfloor - \frac{\beta - V_{\min}}{\varsigma_n} \right) \\ &\in (-\varsigma_n, 0] \end{aligned}$$

Given $x_1, x_2, x_3 \in \mathbb{R}$, we have $[x_1 - x_2]^+ - [x_1 - x_3]^+ \leq [x_3 - x_2]^+$ based on the triangle inequality. The second term yields

$$\begin{aligned} & \frac{1}{1-\alpha}\Big(\mathbb{E}(\big[X-b'\big]^+) - \mathbb{E}([X-\beta]^+)\Big) \\ = \;& \frac{1}{1-\alpha}\mathbb{E}\left(\big[X-b'\big]^+ - [X-\beta]^+\right) \\ \leq \;& \frac{1}{1-\alpha}\mathbb{E}\left(\big[\beta-b'\big]^+\right) \\ = \;& \frac{1}{1-\alpha}\mathbb{E}\left(\left[\beta - V_{\min} - \varsigma_n \cdot \left\lfloor \frac{\beta - V_{\min}}{\varsigma_n} \right\rfloor\right]^+\right) \\ = \;& \frac{\varsigma_n}{1-\alpha}\mathbb{E}\left(\left[\frac{\beta - V_{\min}}{\varsigma_n} - \left\lfloor \frac{\beta - V_{\min}}{\varsigma_n} \right\rfloor\right]^+\right) \\ \in \;& [0, \frac{\varsigma_n}{1-\alpha}) \end{aligned}$$

Thus, it holds that $\mathsf{CVaR}_\alpha(X^{r,\mathbf{F}\,T}_{\mathcal{M},\pi_2}) - \mathsf{CVaR}_\alpha(X^{r,\mathbf{F}\,T}_{\mathcal{M},\pi_1}) = \mathcal{O}(\varsigma_n)$. As ς_n tends to 0 (i.e., $|B|$ increases), π_2 converges to the CVaR optimal policy. □

Finally, we comment on the use of (categorical or quantile) distributional representations and projections for implementing the above. Note that [56, Proposition 5.28] proves that using the distributional Bellman update with a distributional representation/projection combination (categorical or quantile) is guaranteed to converge based on the stride and the number of iterations.

6.3 Experiments

6.3.1 Setup

We built and evaluated a prototype implementation[1] based on PRISM [9], extending its Java explicit-state engine. The main benchmarks used are described in Section 6.3.2 and the experimental results are discussed in the following sections. We focus initially on solving MDP model using the DVI methods of Sections 6.2.2 and 6.2.3, then evaluate the resulting policies using the DTMC method (Section 6.2.1). We then further evaluate the DTMC method on a set of DTMC models from the PRISM Benchmark Suite [8], in particular comparing this (forward, exact) approach to an alternative solution obtained via DVI. Finally, we compare our risk-sensitive method to the CVaR techniques from [50], using the benchmarks from that paper. All experiments were run on a machine with an AMD Ryzen 7 CPU and 14 GB of RAM allocated to the JVM. We set $V_{\min} = 0$ for all case studies; $V_{\max}$ varies, as detailed below.

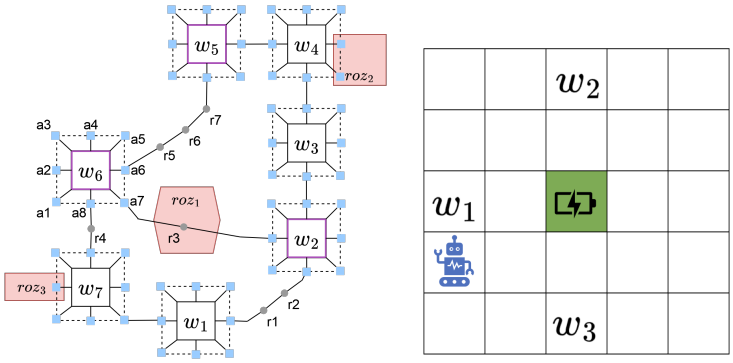
6.3.2 Case Studies

Betting Game. This case study is taken from [51]. The MDP models an agent with an amount of money, initially set to 5, which can repeatedly place a bet of amount $\lambda \in \{0, 1, 2, 3, 4, 5\}$. The probability of winning a bet is 0.7, the probability of losing a bet is 0.25, and the probability of hitting a jackpot (i.e., winning 10λ) is 0.05. The game ends after 10 stages. The reward function is given by the maximal allowance (e.g., 100) minus the final amount of money that the agent owns. We use $V_{\max} = 100$.

Deep Sea Treasure. This case study is also taken from [51]. The model represents a submarine exploring an area to collect one of several treasures. At each time step, the agent chooses to move to a neighbouring location; it succeeds with probability 0.6, otherwise moves to another adjacent location with probability 0.2. The agent stops when it finds a treasure or has explored for 15 steps. The reward function is defined based on the travel cost (e.g., 5 per step) and opportunity cost (i.e., maximal treasure value minus collected treasure value). We set $V_{\max} = 800$.

Obstacle. This case study is inspired by the gridworld navigation example in [49]. We consider an MDP model of an $N \times N$ gridworld with a set of scattered obstacles. The agent's goal is to navigate to a destination, while avoiding obstacles which would cause a delay. At each time step, the agent moves in a selected direction with probability 0.9 and an unintended direction with probability 0.1. The reward function is given by the time spent to reach the destination. We use $V_{\max} = 600$.

[1] Code and models are at https://www.prismmodelchecker.org/files/nfm24dpmc.



(A) UAV (B) Energy

FIGURE 6.2: Maps used in the UAV and Energy case studies.

Human-UAV Interaction. This case study is adapted from the MDP model of the interaction between a human and an unmanned aerial vehicle (UAV) from [105]. A UAV performs road network surveillance missions with the assistance of a human operator. The road network is shown in Figure 6.2a, and the UAV is given a mission specified with LTL formula $\psi = (\mathtt{F}\ w_2) \wedge (\mathtt{F}\ w_5) \wedge (\mathtt{F}\ w_6)$, which translates into covering waypoints w_2, w_5 and w_6 in any order. The reward function is given by the mission completion time. We pick $V_{\max} = 500$.

Energy. This case study considers a robot navigating an $N \times N$ gridworld with energy constraints (Figure 6.2b shows an example for $N = 5$). At each time step, the robot moves to an adjacent grid location with probability 0.7 or ends up in an unintended adjacent location with probability 0.3. The robot starts with a fixed amount of energy and consumes 1 unit per step. The robot can only recharge its battery in the charging station. When the energy is depleted, the robot is transported with a delay to the charging station. The robot is asked to complete a mission specified with LTL formula $\psi = (\mathtt{F}\ w_1) \wedge (\mathtt{F}\ w_2) \wedge (\mathtt{F}\ w_3)$. The reward function represents the mission completion time. We use $V_{\max} = 500$.

6.3.3 Method comparison

Table 6.1 summarizes our experimental results across the benchmarks described above. For each MDP, we run both the risk-neutral and risk-sensitive variants of distributional value iteration (DVI), optimizing expected value and CVaR, as described in Section 6.2.2 and Section 6.2.3, respectively. For the risk neutral case we also run standard value iteration (VI), as implemented in PRISM. For all three methods, we then evaluate the resulting policy, computing the full reward distribution using the forward distribution generation method described in Section 6.2.1, allowing us to compute more precise results for the expected value and CVaR on those policies.

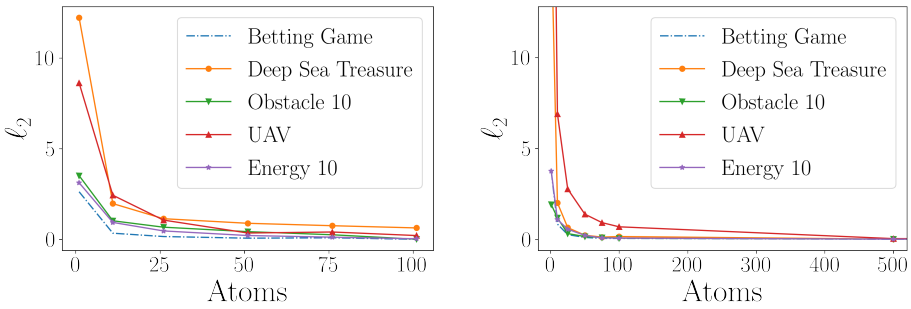
TABLE 6.1: Experimental results: Timing and accuracy of each method.

Model	Method	MDP	Time (s)	$\mathbb{E}$	CVaR_α	$\text{Time}_{\mathsf{dtmc}}$ (s)	$\Delta^{\%}_{\mathbb{E}}$	$\Delta^{\%}_{\mathsf{CVaR}}$
Betting Game	risk-neut. VI	$8.9 \cdot 10^2$	< 1	**61.9**	-	< 1	-	-
	risk-neut. DVI	$8.9 \cdot 10^2$	< 1	**61.9**	98.0	< 1	0.0	0.0
	risk-sens. DVI	$9.0 \cdot 10^4$	36	85.3	**92.2**	< 1	0.0	0.0
DS Treasure	risk-neut. VI	$1.2 \cdot 10^3$	< 1	**359.3**	-	< 1	-	-
	risk-neut. DVI	$1.2 \cdot 10^3$	< 1	**359.3**	474.6	< 1	0.0	0.33
	risk-sens. DVI	$1.2 \cdot 10^5$	72	370.1	**458.6**	< 1	0.0	0.32
Obstacle ($N = 150$)	risk-neut. VI	$2.3 \cdot 10^4$	< 1	**402.8**	-	1,838	-	-
	risk-neut. DVI	$2.3 \cdot 10^4$	97	**402.7**	479.2	1,838	0.01	1.95
	risk-sens. DVI	$2.3 \cdot 10^6$	15,051	402.9	**478.4**	1,673	0.01	2.00
UAV	risk-neut. VI	$1.7 \cdot 10^4$	< 1	**124.1**	-	< 1	-	-
	risk-neut. DVI	$1.7 \cdot 10^4$	4	**123.8**	168.8	< 1	0.2	0.47
	risk-sens. DVI	$1.7 \cdot 10^6$	2,366	134.9	**169.1**	< 1	0.0	0.01
Energy ($N = 15$)	risk-neut. VI	$2.6 \cdot 10^4$	10	**184.3**	-	251	-	-
	risk-neut. DVI	$2.6 \cdot 10^4$	108	**184.0**	382.0	234	0.17	0.47
	risk-sens. DVI	$1.3 \cdot 10^6$	9,384	184.6	**380.9**	122	0.16	0.33

The table shows the time to run each algorithm and the values computed during optimization (the value for the objective being optimized is shown in bold). Additionally, the table shows the time to run the forward distribution method on the induced DTMC, and the (percentage) relative error when comparing the VI/DVI results with the forward distribution outcomes.

For each case study, we also report the number of states in the (product) MDP that is solved. The UAV and Energy benchmarks use non-trivial co-safe LTL formulae for the mission specification (the others are reachability specifications) and so the MDP is a MDP-DFA product. For risk-sensitive DVI, the state space is also augmented with a slack variable resulting in larger product MDPs. We set the slack variable size to $|B| = 51$ for the Energy model, and $|B| = 101$ for the rest. We use the the categorical representation with $m = 201$ for DVI, with $\epsilon = 0.01$ for the convergence metric. For policy evaluation, we use precision $\varepsilon = 10^{-3}$ for the Obstacle and Energy case studies and $\varepsilon = 10^{-5}$ for the others.

Our DVI methods successively optimize their respective objectives on a range of large MDPs. Generally, the policy resulting from the risk-neutral method has a lower expected value, while the policy from the risk-sensitive method has a lower CVaR_α, and the risk-neutral method yields the same optimal policy as baseline VI. As expected, DVI methods are more expensive than VI, since they work with distributions, but the DVI methods are successfully applied to MDPs with several million states. Additionally, the baseline VI method can only provide expected reward values, while the distribution returned by



(A) Categorical Rep. (B) Quantile Rep.

FIGURE 6.3: Experimental results for variable number of atoms in DVI.

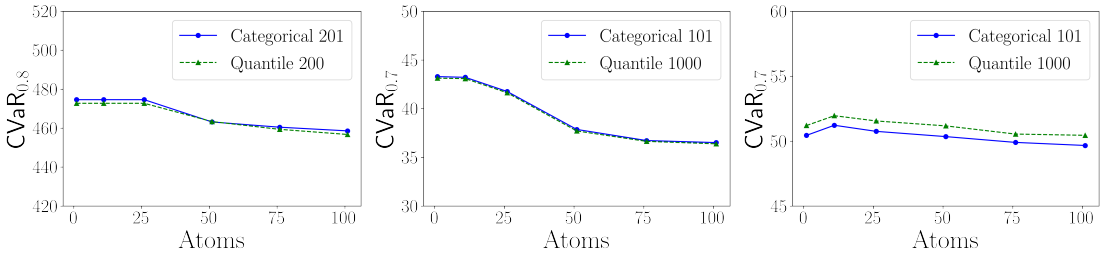
our methods can be used to compute additional distributional properties (variance, VaR, etc.). Comparing the two variants of DVI, the risk-sensitive version takes considerably longer to run. This is primarily due to the use of a larger product model, incorporating a slack variable, rather than the computation required for DVI itself. For the same reason, risk-neutral DVI scales to larger models, but for clarity Table 6.1 only includes models that all methods can solve.

The DTMC forward computation also works on all models. It is often very fast (under a second in 3 cases), but grows expensive on models where the support of the distribution is large. From its results, we see that both DVI methods produce approximate distributions that are close to the true distribution.

Note that in the last three case studies, the $V_{\max}$ value is higher, resulting in a larger stride and thus more coarse representations for both the value distributions and the slack variable (for risk-sensitive DVI). This results in more approximation errors when computing metrics from the value distributions generated using DVI. This can be seen in the case of the UAV model where the risk-neutral method underestimates CVaR_α (168.8 compared to 169.6 from the true distribution generated by the DTMC method for the same policy). The following experiments aim to evaluate how the parameters of the distributional representation affect the resulting approximate distributions generated by DVI.

6.3.4 Effects of using discrete distribution representations

Approximation of the return distribution. Figures 6.3a and 6.3b plot the effects of varying the number of atoms in categorical and quantile representations, in terms of the ℓ_2 distance between the approximate distribution resulting from the risk-neutral DVI and the ground truth (obtained via applying the DTMC forward distribution generation method with $\varepsilon = 10^{-5}$ on the resulting optimal policy). *For both representations, the ℓ_2 distance approaches 0 as the number of atoms increases, indicating that the approximate*



(A) Deep Sea Treasure (B) Obstacle 10 (C) Energy 10

FIGURE 6.4: Experimental results of risk-sensitive DVI.

distributions become very close to the ground truth. We observe similar effects with the risk-sensitive method and thus omit the resulting plot.

A larger number of atoms (m value) leads to a higher computational cost, thus we consider smaller models for the Obstacle and Energy case studies with $N = 10$ for plotting. As an illustration of accuracy/cost trade-off, for Energy 10, the runtime using categorical representations with 11 atoms (resp. 101 atoms) is 0.3s (resp. 0.63s), while the runtime when using quantile representations with 10 atoms (resp. 100 atoms) is 0.9s (resp. 5s). The quantile projection is more expensive than the categorical projection, resulting in higher runtimes.

Approximation of the slack variable. Figure 6.4 illustrates the effects of varying the number of atoms used in slack variables ($|B|$) in risk-sensitive DVI. *The results show that increasing $|B|$ generally leads to better policies with smaller* CVaR *values.* This is in part because the algorithm would check a larger set of initial risk budgets $\bar{b} \in B$. But there is a trade-off since the computational cost grows with an increasing $|B|$. For example, in the Energy 10 model, the runtime using the categorical representation with 101 atoms for $|B| = 11$ (resp. $|B| = 101$) is 7.8s (resp. 78.6s), whereas the runtime of using the quantile representation with 1,000 atoms for $|B| = 11$ (resp. $|B| = 101$) is 477s (resp. 5,163s).

6.3.5 DTMC Performance Analysis

Next, we further evaluate the forward computation method for DTMCs from Section 6.2.1 on a range of common DTMC benchmarks from the PRISM benchmark suite [8]. In particular, we compare to an alternative computation using the risk-neutral DVI method method of Section 6.2.2, treating DTMCs as a special case of MDPs. Table 6.2 shows the performance of the two methods. For each model, we indicate the parameters used for the benchmark and the DTMC size (states and transitions). For the DVI method, we use the categorical representation with a stride of 1 and a value of $V_{\max}$ large enough to represent the distribution (also shown in the table).

TABLE 6.2: Performance comparison for DTMC forward computation

Model	Param.s	States	Transitions	$V_{\max}$	DVI(s)	DTMC(s)	$\Delta^{\%}_{\mathbb{E}}$	$\Delta^{\%}_{\mathsf{CVaR}}$
EGL	N=8,L=3	$5.4 \cdot 10^6$	$5.5 \cdot 10^6$	40	439	1	0.4	0.5
	N=8,L=4	$7.5 \cdot 10^6$	$7.6 \cdot 10^6$	40	897	1	0.4	0.4
	N=8,L=5	$9.6 \cdot 10^6$	$9.7 \cdot 10^6$	50	4,345	1	0.3	0.4
Leader	N=8,K=5	$2.7 \cdot 10^6$	$3.1 \cdot 10^6$	20	41	2	0.0	0.0
	N=10,K=4	$9.4 \cdot 10^6$	$1.0 \cdot 10^7$	30	577	15	0.3	0.6
	N=8,K=6	$1.2 \cdot 10^7$	$1.3 \cdot 10^7$	20	163	9	0.0	0.1
Herman	N=13	$8.2 \cdot 10^3$	$1.6 \cdot 10^6$	100	4	14	0.6	0.9
	N=15	$3.3 \cdot 10^4$	$1.4 \cdot 10^7$	120	57	190	0.5	0.9
	N=17	$1.3 \cdot 10^5$	$1.3 \cdot 10^8$	140	1,234	2,369	0.8	1.2

In two out of the three benchmarks, the DTMC forward computation is much faster. This is because the DVI method calculates a reward distribution for every state of the model. However, for the third example, where $V_{\max}$ is significantly higher, DVI is actually faster (the same can be seen for the Obstacle and Energy models in Table 6.1). The DTMC method computes distribution to a pre-specified accuracy, but DVI may incur approximation errors, primarily due to convergence. The (relative) errors for the expected value and CVaR metrics are also shown for every benchmark in Table 6.2.

6.3.6 Performance comparison with risk-aware SSP

Lastly, we compare our risk-sensitive DVI method with the "risk-aware SSP" (stochastic shortest path) methods of MDP CVaR optimisation from [50], which presents both a linear programming (LP) and a VI approach. Note that these are CVaR-specific, whereas our approach generates the full reward distribution, allowing for the subsequent computation of other distributional properties. Moreover, the implementation of [50] focuses on the case where the reward for every step is one, compared to the more general reward functions we support and implement. Hence, we cannot compare using the models from Table 6.1 and instead use the benchmarks from [50].

Table 6.3 shows the time for the LP and VI methods of [50] and the time to run our risk-sensitive DVI method (Section 6.2.3) followed by the DTMC method (Section 6.2.1). We use the same values of α (correctly resulting in the same CVaR_α values) and the same timeout of 1 hour. Considering VI (the better performing of the two risk-aware SSP methods), we see that it is significantly faster than our method on the FireWire and WLAN benchmarks, at least in part because of the larger state space resulting in more distributions to maintain. However, for the Grid example, as the model size increases, our approach is faster. In terms of atoms, we used $|B| = m = 51$, $|B| = m = 61$ and $|B| = m = 41$ for the Grid, FireWire, WLAN models respectively.

TABLE 6.3: Performance comparison with risk-aware SSP

Model	States	Transitions	SSP-LP (s)	SSP-VI (s)	Ours (s)
Grid ($x = 4$)	$1.3 \cdot 10^3$	$1.2 \cdot 10^4$	2	0	8
Grid ($x = 8$)	$3.2 \cdot 10^3$	$3.8 \cdot 10^4$	159	1	19
Grid ($x = 16$)	$6.4 \cdot 10^3$	$8.6 \cdot 10^4$	1,915	6	46
Grid ($x = 32$)	$1.3 \cdot 10^4$	$1.8 \cdot 10^5$	timeout	143	101
FireWire	$1.4 \cdot 10^5$	$1.8 \cdot 10^5$	memout	2	920
WLAN	$8.7 \cdot 10^4$	$3.0 \cdot 10^5$	memout	1	550

6.4 Summary

In this chapter, we present a distributional probabilistic model checking approach, which supports a rich set of distributional queries for DTMCs and MDPs. Experiments on a range of benchmark case studies demonstrate that our approach can be successfully applied to check various distributional properties (e.g., CVaR, VaR, variances) of large MDP and DTMC models. The instances used in the experimental section are tailored to be approachable by both Distributional VI methods but results show that the risk-sensitive algorithm is significantly less scalable. One way of improving the scalability of both methods would be to use dynamic $V_{\min}$ and $V_{\max}$ values since not all states of the MDP require the full range of values. Another would be to adapt representations such as implicit quantile networks [106] to be used for the value distributions at the cost of slightly less interpretable distributions.

We believe that the work proposed in this chapter paves the way for applying distributional probabilistic model checking in many safety-critical and risk-averse domains (e.g., human-robot interaction, autonomous vehicles). In addition, while the risk-neutral DVI method does not synthesize a risk-aware policy, it is efficient for even larger models than those presented in this chapter and can still be used to evaluate a policy's performance in terms of risk metrics. Furthermore, when DTMC models that are too large for the method presented in Section 6.2.1, the risk-neutral method can be used as an efficient approximation of the distribution. This allows the designer of the system to use our approach with larger models as well as provide additional safety insights into policies obtained by our approach or other alternatives. We believe that increased understanding of policy mechanics provided by our approaches can be beneficial in additional areas of model checking such as multi-agent or partially observable environments. Many of those domains feature unique challenges, some with increased risk of undesirable outcomes. In the next chapter, we explore how to provide a distributional outlook on probabilistic model checking in uncertain environments.

Chapter 7

Distributionally Uncertain Probabilistic Model Checking

In the previous chapter, we proposed a general framework for assessing a variety of distributional metrics such as risk-specific values. However, requiring full information about the transitions of the MDP models to be evaluated is impractical in some cases. In addition, using only the mean of the transition probabilities to construct the MDP or DTMC can lead to flawed planning and safety analysis. Previous work on parametric MDPs have shown how uncertainty in the parametric transition probabilities of the model lead to a large set of possible solutions [73]. Moreover, few methods provide ways to tractably represent and assess the effects of this uncertainty.

In this chapter, we turn our attention to policy synthesis and evaluation for uncertain parametric MDPs. We seek to answer the following questions. How can we leverage the information gathered about parametric uncertainty to synthesize a balanced policy? How can we give the designer interpretable results showing the uncertainty's effects on the obtained policy?

Recent extensions to probabilistic model checking propose ways to represent uncertain models and handle partially known transition systems [73], [74]. These techniques support the synthesis of robust policies and verification against temporal logic specifications some of which are implemented in popular model checkers such as PRISM [9] and Storm [10]. Typically, variations of parametric Markov decision processes (MDPs) such as interval MDPs [79] and uncertain parametric MDPs [74] are used. In addition, methods supporting corresponding variations of discrete-time Markov chains (DTMCs) have also been developed for verification and policy evaluation [73]. These models are uncertain and parametric in the sense that they use variable parameters to represent uncertainty.

Rewards (or costs) are commonly used to express the values associated with events which can be minimized (or maximized) during optimization as well as for quantitative verification. Relevant examples include checking the minimum expected time for a set of

messages to be successfully sent with a given network protocol, or finding a policy that maximizes the amount of money won after a number of gambling rounds. The use of standard MDPs usually assumes that the designer has complete information about the probabilistic behavior which can be unrealistic. Data used to model the system might be based on sampled averages which do not reflect the complete behavior. For example, when playing a slot machine, each machine will have different probability distributions for a win. But optimizing based on sampled averages transition functions may lead to misleading results. Developing methods that can handle a more detailed characterization of the data to synthesize more robust policies is essential for safety-critical applications.

Interval MDPs mitigate this by using a parameter interval to describe the transition function [75], [80]. However, approaches targeting Interval MDPs tend to focus on exclusively the best or worst case resulting in overly conservative policies. Distributionally-Robust MDPs (DR-MDPs) address this by using a distribution over the transition functions [65], [107]. Previously proposed methods with DR-MDPs either suffer from intractability or lack of expressiveness because of their representation of uncertainty.

In this chapter, we leverage uncertain parametric MDPs (upMDPs), where we express the uncertain parameters using discrete distributional representations. In addition, we design a tractable algorithm to optimize the weighted expected value. Rather than computing a scalar value to represent the expected reward for each upMDP state, we return a distribution of expected accumulated rewards for each state. Specifically, following previous work from [56] and Chapter 6, we consider the categorical and quantile representation for both the distribution of rewards and for the joint distribution of uncertain parameters. To further evaluate our results, we construct the induced DTMC for each of the parameter realizations and generate a precise distribution using the DTMC method presented in the previous chapter. Finally, we implement a prototype implementation as an extension of PRISM [9] and evaluate its performance on a range of case studies.

7.1 Background

Background relating to MDP, DTMC and distributional representations can be found in Chapter 3. In this section, we focus on relevant definitions for parametric models.

Definition 9 (pDTMC)**.** A parametric discrete-time Markov chain (pDTMC) is a tuple $\mathcal{D}_p = (S, s_0, P, AP, L, \mathbb{T})$, where S is a set of states, $s_0 \in S$ is an initial state, $\mathbb{T}$ is a finite set of admissible parameter values, $P : S \times S \times \mathbb{T} \to [0, 1]$ is a probabilistic transition matrix satisfying $\forall s \in S : \sum_{s' \in S} P(s, s') = 1$, AP is a set of atomic propositions and $L : S \to 2^{AP}$ is a labelling function.

A pDTMC $\mathcal{D}_p$ evolves similarly to a DTMC, starting in s_0, and the probability of taking a transition from s to s' is $P(s, s')$ for an instantiation $\tau \in \mathbb{T}$ where $\tau \in \mathbb{R}$.

Definition 10 (pMDP)**.** A parametric Markov Decision Process (pMDP) is a tuple $\mathcal{M}_p = (S, s_0, A, P, AP, L, \mathbb{T})$ where S is a set of states, $s_0 \in S$ is an initial state, atomic propositions AP, the parameter set $\mathbb{T}$ and labelling function L defined similarly as for a pDTMC, and $P : S \times A \times S \times \mathbb{T} \to [0, 1]$ is a probabilistic transition function satisfying $\forall s \in S, \forall a \in A : \sum_{s' \in S} P(s, a, s') \in \{0, 1\}$ for a given $\tau \in \mathbb{T}$.

Similarly to MDPs, in each state s of a parametric MDP $\mathcal{M}_p$, there are one or more available actions which can be taken, denoted $A(s) = \{a \in A \mid P(s, a, s') > 0 \text{ for some } s'\}$. For any admissible instantiation $\tau \in \mathbb{T}$, we obtain an MDP $\mathcal{M}_p[\tau]$. At each state, the choice of actions at a state $s \in S$ is determined by a policy $\pi(s) \to A(s)$. For the targets considered in this chapter, we focus on memoryless policies for which we can build a (finite) *induced pDTMC* which is equivalent to $\mathcal{M}_p$ acting under π.

Definition 11 (Reward structure)**.** A *reward structure* is, for a pDTMC $\mathcal{D}_p$, a function $r : S \to \mathbb{N}$ and, for an pMDP $\mathcal{M}_p$, a function $r : S \times A \to \mathbb{N}$.

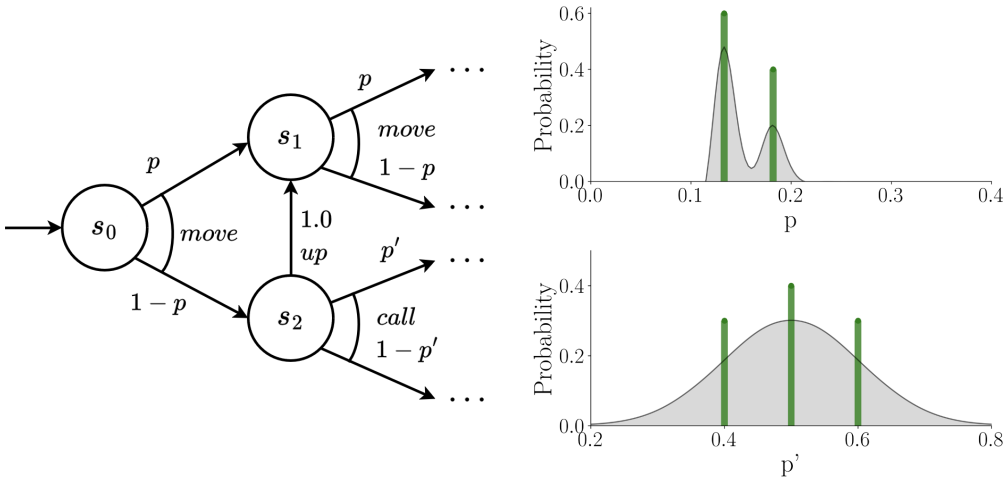
We augment both pDTMC and pMDP models with rewards structures which associate a numerical value to certain states or transitions. For an infinite path ω, we also write $r(\omega, k)$ for the sum of the reward values over the first k steps of the path, i.e., $r(s_0 s_1 s_2 \dots) = \sum_{i=0}^{k-1} r(s_i)$ for a DTMC and $r(s_0 a_0 s_1 a_1 s_2 \dots) = \sum_{i=0}^{k-1} r(s_i, a_i)$ for an MDP. Note that while the rewards do not depend on the uncertain parameter, the probability of receiving the rewards does. In other words, the rewards themselves are independent from the uncertain parameters but the expected rewards at a given state depends on the uncertain transitions affected by the aforementioned parameters.

7.2 Problem Formulation

We base our approach on probabilistic model checking, which is a widely used framework for formally analyzing properties of uncertain models. We first introduce the problem formulation for this chapter.

Definition 12 (upMDP [74])**.** An uncertain parametric Markov Decision Process (upMDP) is a tuple $\mathcal{M}_{\mathbb{D}} = (S, s_0, A, P, AP, L, \mathbb{T}, \mathbb{U})$ where S is a set of states, $s_0 \in S$ is an initial state, atomic propositions AP, the parameter set $\mathbb{T}$ and labelling function L defined similarly as for a pMDP, and $\mathbb{U}$ is a probability distribution over the parameter space $\mathbb{T}$. For any instantiation of the uncertain parameters $\tau \in \mathbb{T}$ where $P(\tau|\mathbb{U}) > 0$, we have a concrete MDP $\mathcal{M}_{\mathbb{D}}[\tau]$.

When there are multiple uncertain parameters ($|\mathbb{T}| > 0$), τ takes the form of a tuple with $\tau \in \mathbb{R}^n$. In order to effectively handle uncertain parametric models in our computer, we



(A) An example upMDP $\mathcal{M}_{\mathbb{D}}$

(B) Example categorical representations for p and p' with $k = 2$ and $k = 3$ atoms resp.

require the use of discrete distributional representation. Specifically, we implement the categorical and quantile representations described in Definitions 1 and 2 for the reasons discussed in the previous chapter and in [56]. Since the representations used are finite, a dDTMC can be induced from a upMDP $\mathcal{M}_{\mathbb{D}}$, a policy π and a parameter combination τ. For clarity, we distinguish the number of atoms in the input distribution k from the atoms used in the value distribution representation m.

The core property we consider is the expected rewards (or costs) that are accumulated until a target is reached (eg. completing a task, or successfully sending messages over a network). To reason about the accumulation of rewards, we define the reward structure for a upMDP similarly to the reward structure for MDPs in Definition 11. The different possible expected returns stem from the different possible combinations of parameter values based on the finite representations used. We focus on the setting where the choice of the parameter values is *uncontrollable*. Intuitively, we are modeling the case where the parameters model represent uncontrollable phenomenons (wind direction, rain, etc.) or we are interested in solving the problem for a given amount of information obtained from collected data.

Problem Statement. Given a upMDP $\mathcal{M}_{\mathbb{D}}$ and a target set $T \subseteq S$, we seek to optimize the *weighted* expected value of the distribution of possible expected accumulated rewards. We exploit the probability distributions from $\mathbb{U}$ over the parameter space to compute the weighted expected value and factor it into our policy selection. Furthermore, we return an approximation of the distribution of possible expected rewards using a distributional representation for policy evaluation.

Without loss of generality, we focus on the minimization case for the remainder of this

chapter. Moreover, our approach can directly be extended to more general LTL specifications by using a product upMDP as is done in Chapter 6.

Example 5.[Motivating Example] We consider a ground robot on a search and rescue mission after an earthquake influenced by two sources of uncertainty. The first (p) represents the uncertainty in the motion of the robot due to lack of information about how the terrain may have shifted. The second (p') expresses possible perturbations of the cellular network. In Figure 7.1a, we show a fragment of a possible upMDP for this scenario. Note that some transitions can be purely probabilistic and independent of the parameter as seen for the "up" action. The uncertain parameter p in this example influences the transitions for the "move" action at states s_0 and s_1. Meanwhile, parameter p' affects the success of the "call" action at state s_2.

In Figure 7.1b we show the parameters' true distribution (shaded gray) and the corresponding categorical representations (Definition 1). Parameter p's true distribution is multi-modal while p' follows a normal distribution with a mean of 0.5. The green bars represent how the continuous underlying distributions are discretized into their distributional representations.

To compute the distribution for the transition associated with the function $f(p) = 1 - p$, we evaluate $1 - p_i$ where p_i is each support of the distribution for p. The probability δ_{θ_i} (0.6 or 0.4) of a support atom i becomes the probability of $1 - p_i$. For atom $i = 0$ which occurs with probability 0.6, the action "move" has a probability $1 - 0.14$ to transition to s_1. The same evaluation method applies to parameter p'. Our framework allows for the definition of general functions of the uncertain parameter which are evaluated during optimization. ■

7.3 Approach

In this section, we describe our method for computing the distribution of expected cumulative rewards of an upMDP $\mathcal{M}_{\mathbb{D}}$ under a policy optimizing for the average case of the input parameter distributions $\mathbb{U}$. We also refer to the distribution of expected cumulative rewards as the value distribution for a given state. We focus on probabilistic reachability of a target set $T \in S$. Following standard practice for probabilistic model checking, we assume that there exists an optimal policy with finite expected rewards which reaches the target set with probability 1 for each realization of the uncertain parameters. This can be checked as a preliminary step by a graph analysis of every instantiated MDP $\mathcal{M}_{\mathbb{D}}[\tau]$ for each $\tau \in \mathbb{U}$ [102].

The pseudocode for our algorithm is shown in Algorithm 7 which aims to provide a policy that exploits the probability distributions of the uncertain parameters. The first step is to

Algorithm 7: Distributionally Uncertain Value Iteration

Input : upMDP $\mathcal{M}_{\mathbb{D}}$, $|\mathbb{T}|$ parameter distributions, reward structure r, target set $T \subseteq S$, and convergence threshold $\epsilon \in \mathbb{R}_{>0}$

Output: optimal policy π^*, value distribution μ_{s_0} under π^*

1 Construct joint distribution $\mathbb{U}$
2 **foreach** $s \in S$ **do**
3 $\quad \mu_s \leftarrow \delta_0$
4 **while** $e > \epsilon$ **do**
5 $\quad$ **foreach** $s \in S \setminus T$ **do**
6 $\quad\quad$ **foreach** $a \in A(s)$ **do**
7 $\quad\quad\quad$ **foreach** $\tau \in \mathbb{U}$ **do**
$\quad\quad\quad\quad$ `// Evaluate transition function`
8 $\quad\quad\quad\quad P^\tau = \texttt{eval}(\mathcal{M}_{\mathbb{D}}, \tau)$
$\quad\quad\quad\quad$ `// Compute weighted particles`
9 $\quad\quad\quad\quad \text{particles}(\tau) = \texttt{tuple}\left(P(\tau|\mathbb{U}),\ r(s,a) + \sum_{s' \in S} P^\tau(s,a,s',\tau) \cdot \mu_{s'}\right)$
10 $\quad\quad\quad \eta(s,a) :\overset{D}{=} \texttt{proj}\ (\text{particles})$
11 $\quad\quad \pi^*(s) \leftarrow \arg\min_{a \in A(s)} \mathbb{E}\big(X | X \sim \eta(s,a)\big)$
12 $\quad\quad \mu'_s \leftarrow \eta(s, \pi^*(s))$
13 $\quad e \leftarrow \sup_{s \in S \setminus T} d(\mu_s, \mu'_s)$
14 $\quad$ **foreach** $s \in S \setminus T$ **do**
15 $\quad\quad \mu_s \leftarrow \mu'_s$
16 **return** π^* and μ_{s_0}

construct the joint distribution $\mathbb{U}$ using the distributional representations for each of the uncertain parameters in $|\mathbb{T}|$. For the purpose of this work, we assume that the designer has already chosen an appropriate distributional representation for each parameter. Thus, the joint probability distribution represents each possible parameter's support combination such that the support has a non-zero probability. We then initialize the distribution of expected reward to δ_0 for each state $s \in S$. In lines 5-12, the algorithm loops through the non-target states and updates the distribution of expected rewards as follows.

To find the optimal action for each state, we iterate over the available actions $A(s)$ and compute the value distribution which we use to compare the expected values of the different actions in line 11. To compute the value distributions, we first need to compute an intermediate distribution based on the successor states and possible transition functions. The transition function P is evaluated with a specific $\tau \in \mathbb{U}(s,a)$ in line 8. This intermediate distribution consists of a set supports and associated probabilities which we call "particles". Each particle's support corresponds to the expected value conditioned by a parameter combination realization. The probability of a particle is the probability of the parameter combination instance under the joint distribution $\mathbb{U}$.

In line 9, the support is the sum of the reward for the current state in addition to the expected value of the rewards for each successor state $s' \in S$ states. However, for the particles to be saved efficiently, we need to project them into the chosen distributional representation for the value distributions. We choose the optimal action based on the expected value of the value distributions of the admissible actions (line 11). The value distribution corresponding to the optimal action is saved for the appropriate state (line 12). We check convergence by computing the maximum change in the value distribution at each state from an iteration to the next (line 13). This change is calculated in the form of a distributional distance measure such as the Wasserstein distance or the Cramér distance (see Section 3.2.2). The choice of distance measure depends on the chosen distributional representation (see discussion in Chapter 6).

Example 6. Figure 7.1b shows the parameter distributions for example 5. The supports of the joint distribution (or parameter realizations) are:

$$\mathbb{U} = \{(0.14, 0.4)\ (0.14, 0.5)\ (0.14, 0.6)\ (0.18, 0.4)\ (0.18, 0.5)\ (0.18, 0.6)\}$$

For the first possible realization, its associated probability can be computed by multiplying $P(p = 0.14) \cdot P(p' = 0.4)$.

Let us assume that the reward for taking the "move" action at state s_0 is 2, that the value distributions computed at the previous iteration are $\mu(s_1) = \{(10, 20); (0.3, 0.7)\}$ and $\mu(s_2) = \{(5, 10); (0.8, 0.2)\}$. The first tuple represents the supports while the second shows the probabilities.

At line 7, for state s_0 at the current iteration, we would first pick $\tau = (0.14, 0.4)$ and $P(\tau|\mathbb{U}) = 0.18$. The transition function for the *move* action would then have two transitions with probabilities $P(s_0, move, s_1) = 0.14$ and $P(s_0, move, s_1) = 0.86$.

At line 9, we first compute the sum of the successor states values. In this case, the support of the particle would be:

$$2 + [0.14 \cdot (10 \cdot 0.3 + 20 \cdot 0.7)] + [0.86 \cdot (5 \cdot 0.8 + 10 \cdot 0.2)] = 9.54$$

Thus, the first particle has a probability of 0.18 and a support of 9.54. A particle is calculated for each of the possible parameter realizations. The list of particles is then projected onto a distributional representation. ■

Correctness. In addition to the approximation introduced by using the distributional representation, this method computes the expected value using all possible parameter realizations for the successor states for every given parameter combination. In other

words, the expected value computed for a given τ uses the values from the successor states computed including τ'. In our motivating example, it is unrealistic to consider the results for the combination $(p = 0.14, p' = 0.6)$ from state s_1 when calculating the expected value for the combination $(p = 0.18, p' = 0.6)$ at state s_0. Intuitively, when instantiating MDP $\mathcal{M}_{\mathbb{D}}[\tau]$ the combination of parameter values τ are consistent throughout. For this reason, the values reported in the experiments are refined by computing expected value of the distribution of rewards for the induced DTMCs using the distribution generation method from Chapter 6. The DTMCs are induced using the policy synthesized with Algorithm 7 and every MDP instantiation $\mathcal{M}_{\mathbb{D}}[\tau]$ for $\tau \in \mathbb{U}$.

7.4 Experiments

7.4.1 Setup

We built and evaluated a prototype implementation[1] based on PRISM [9], extending its Java explicit-state engine. The main benchmarks used are described in Section 7.4.2 and the experimental results are discussed in the following sections. We initially focus on comparing the policies obtained using the method in Section 7.3 for the MDP model and the policy from Robust VI. Then we analyze the effects of using a discrete distributional representation for the distribution of the uncertain parameters. All confidence intervals computed for the experiments use a confidence level of 0.8. All experiments were run on a machine with an AMD Ryzen 9 CPU and 14 GB of RAM allocated to the JVM.

7.4.2 Case Studies

Betting Game. This case study is adapted from [58]. The MDP models an agent with an amount of money, initially set to 5, which can repeatedly place a bet of amount $\lambda \in \{0, 1, 2, 3, 4, 5\}$. This model features two uncertain parameters, the probability of winning a bet which is within $[0.3, 0.7]$ and the probability of hitting a jackpot (i.e., winning 10λ) which is within $[0.05, 0.15]$. The probability of losing a bet is the remaining probability based on the joint parameter realization. The game ends after 10 stages. The reward function is given by the maximal allowance (e.g., 100) minus the final amount of money that the agent owns.

Drone. This case study is taken [75], models an $N \times N \times N$ aerial world with several obstacles where the objective is to reach a target location $(N-1, N-1, N-1) \pm 1$ opposite from the starting location (2,2,2). The uncertainty in this model is area-based to represent the different weather conditions in different areas. The first parameter $\in [0.2, 0.7]$ regulates the strength of the winds pushing the drone in the $-x$ direction. The

[1] Code and models are available from https://github.com/davexparker/prism/tree/dpmc.

Model Information						Metrics				
Name	Instance	States	Transitions	$\lvert\mathbb{T}\rvert$	Method	$\lvert\mathbb{U}\rvert$	Value	Std Dev.	Difference	Time(s)
Consensus	N=2,K=2	272	492	2	Robust VI	-	339.2	-	641.3	0.3
					upMDP	42	50.1	43.9	**79.1**	17.6
Consensus	N=2,K=4	528	972	2	Robust VI	-	864.6	-	1,648.8	6.6
					upMDP	42	195.2	228.9	**588.9**	77.5
Betting Game	STAGES=10	891	10,740	2	Robust VI	-	60.1	-	65.0	0.2
					upMDP	20	58.0	19.4	**60.0**	4.1
Drone	N=9	1,744	11,884	2	Robust VI	-	59.4	-	64.4	0.8
					upMDP	36	52.8	18.9	**47.2**	271.8
Firewire	delay=3	4,093	5,583	1	Robust VI	-	137.0	-	**37.4**	0.8
					upMDP	7	138.6	13.3	**37.4**	3.7

TABLE 7.1: Experimental results: timing and policy metrics of each method.

second $\in [0.4, 0.7]$ regulates the strength of the wind pushing the drone in the $-y$ direction. A cost of 1 is accumulated for each time step, hitting an obstacle results in the drone being reset to the start.

Consensus. This benchmark is adapted from [8] and models the randomized consensus protocol of Aspnes and Herlihy where the objective is to achieve agreement for N asynchronous distributed processes (eg. $N = 2$). In each round, the processes check the status others and attempt to find an agreement which involves a distributed random walk. In the case where they do not reach an agreement, a coin is flipped to decided their next choice. The two uncertain parameters (within $[0.2, 0.7]$ and $[0.3, 0.6]$) used for this model represent the probability of flipping the coin to head vs tails for each process. A cost of 1 is accumulated for each time step, we are interested in minimizing the time taken to reach consensus.

Firewire. This case study modeling the FireWire IEEE 1394 root contention protocol is also taken from [8]. This type of protocol is used in distributed systems to elect a root node within network nodes. When a new election may be needed, a coin flip is used to determine whether to wait a short or long delay before checking the network again to see if a node has claimed leadership. The uncertain parameter $\in [0.2, 0.7]$ represents the probability of choosing the shorter delay. The goal is to minimize the amount of time units needed to elect a leader node.

7.4.3 Main results

We report our experimental results in table 7.1 for the benchmarks described above. For each MDP model using the method presented in Section 7.3, then exhaustively evaluate the resulting policy for each value of the joint parameter distribution using the Distributional DTMC method from [13] for more precise results. For the Robust VI baseline, we

first compute confidence bounds over the input distribution which we use as the interval for the IMDP model.

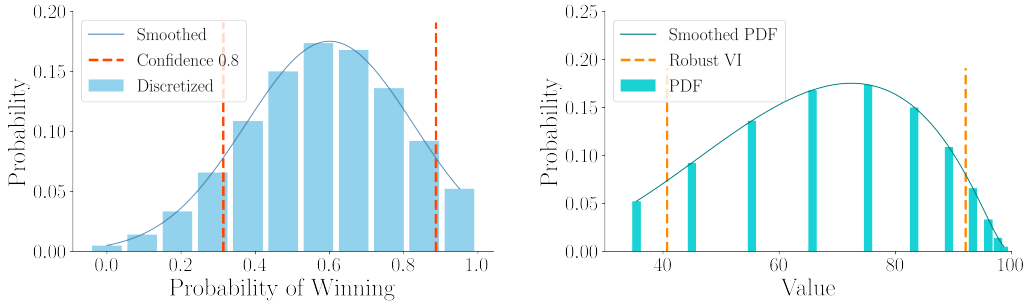
Table 7.1 shows the constant values used for each benchmark as well as model information such as the state and transition spaces. We also show the number of uncertain parameters in $\mathbb{T}$ as well as the size of the joint distribution $\mathbb{U}$ using the distributional representation. The higher the number of parameters and the precision used to represent their associated distributions, the larger the size of the joint distribution. For our method (upMDP), we show the total time to run the method in Section 7.3 and the Distributional DTMC method (with $\varepsilon = 10^{-5}$ for the precision). The time reported for Robust VI is the total time to compute the minimum expected value and the maximum expected value. The bounds of the Robust VI expected values are used to compute the difference and the midpoint is used for the value. For upMDP, we calculate the difference using a confidence bound on the output distribution. We use the the categorical representation for the distribution of returns with $m = V_{\max} + 1$ with $\epsilon = 0.01$ for the convergence metric.

We pick $V_{\max} = 100$ for the Betting Game and Drone models, $V_{\max} = 200$ for Firewire and $V_{\max} = 300$ for Consensus. For the distribution of the input parameters we pick $k = 7$, $V_{\min} = 0.2$ and $V_{\max} = 0.8$ for all case studies except for Betting Game where $k = 17$, $V_{\min} = 0$ and $V_{\max} = 0.8$.

Our method provides a tractable way to factor in parameter uncertainty to synthesize a policy. Generally, the policy from our method (upMDP) has a lower difference reflecting how it helps narrow down the interval of expected behavior. This is because our method can leverage the information contained in the input parameter distributions instead of purely maximizing or minimizing. Note that for the Firewire case study, the difference remains is the same for both methods while the baseline reports a slightly lower value. Recall that the value is computed using the midpoint of the interval MDP bounds and does not represent a real policy but a comparison metric only. The two first rows show two different instances of the consensus protocol. The results show a large standard deviation and an even larger interval since in this benchmark the uncertain probabilities are encountered often and highly affect the range of values possible.

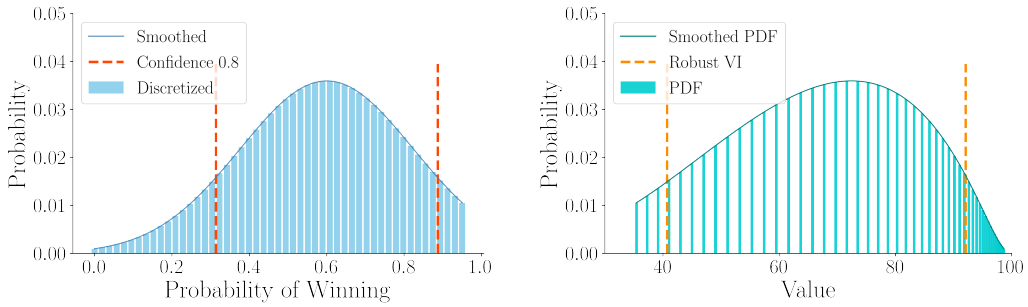
The distribution of expected returns from our method also allows for the computation of additional metrics such as the std deviation. In contrast, the robust VI policies only gives the expected value for the most optimistic and most conservative policies. Using the Distributional DTMC method with the synthesized policy is often very fast but requires more time as the size of the parameters' joint distribution increases and as the transition space increases. This is especially the case for the Drone model where the DTMC method takes 214.1 seconds out of 271.8 total seconds.

7.4.4 Effects of the input distribution approximation



(A) Input distribution with $k = 11$ (B) upMDP

FIGURE 7.2: Output low fidelity



(A) Input distribution with $k = 51$ (B) upMDP

FIGURE 7.3: Output high fidelity

Figures 7.2 and 7.3 plot the effect of varying the precision of the input distribution and the resulting distribution of returns for the Betting Game case study. We fix the probability of hitting the jackpot to 0.05. We use a continuous normal distribution with a mean of 0.6 and variance of 0.05 to model the probability of winning the bet.

In Figure 7.2, we show the results for a less precise (smaller k) approximation of the uncertain parameter distribution. Specifically, we pick $k = 11$ and plot the ensuing discretization as a bar plot in Figure 7.2a. We also plot the confidence bounds for a confidence level of 0.8 which we use for as the intervals in the IMDP model sent as an input to Robust VI. In Figure 7.2b, we plot the output distribution of our method as a bar plot and the lower and upper bounds for Robust VI as dashed lines. Similarly, Figures 7.3a and 7.3b show the same plots for $k = 51$ respectively.

In both setups, our method shows consistent and informative results. While the precision of the approximation for the input parameter is lower in Figure 7.2 than in Figure 7.3, the output distributions are consistent (expected values of 71.09 and 71.73 resp.). We also note that the 0.8 confidence bound for the higher $k = 51$ case ($[47.0, 92.6]$) is tighter

than the lower precision case ($[45.0, 93.3]$). However, the higher precision is more computationally expensive than the lower precision (8.9s and 2.3s resp.). Finding the right trade-off between precision vs time and memory is essential for this method.

7.5 Summary

In this chapter, we design a distributionally uncertain MDP which includes a joint distribution over the uncertain parameters using tractable yet expressive distributional representations. We also propose an algorithm to optimize the weighted expected value of the accumulated rewards over these models. Experiments over a varied set of benchmarks show that it can successfully be applied to medium sized models. Further investigations would be helpful to evaluate our methods performance and precision on larger benchmarks. Another avenue of further research includes exploring how this new method can be adapted to control its sensitivity to risk. We believe the work proposed in this chapter is a first step towards more balanced and informative approaches to uncertain environment model checking.

The main disadvantage of this approach is the lack of formal guarantees for the output distribution. Additional analytical research is required to determine the precision of the approximate method presented in Section 7.3. It would be interesting to develop stronger guarantees in the form of policy performance error bounds relative to the true interval of possible expected rewards. This approach was developed for applications where a large amount of uncertainty is present in the model as is the case when a model is iteratively built as more samples become available. Further experiments in this type of application domain can give a better understanding of our method's applicability in a real-world environment as well as future directions for this research.

Chapter 8

Conclusion

In this dissertation, we proposed several approaches for safer sequential decision-making in uncertain environments. Initially, we addressed safe policies in multi-agent reinforcement learning (MARL) where little information is known about the system and environment. When an abstraction of the environment is available and runtime safety is necessary, we improved the scalability of shielding techniques as presented in Chapter 4. For soft safety requirements, we advanced logic-based reward shaping for MARL which guides the training towards safe behaviors in Chapter 5. However, in certain situations unsafe states are unavoidable because of a non-deterministic and probabilistic environment. In this case, probabilistic model checking offers a strong foundation for in-depth analysis. We designed a distributional framework for probabilistic model checking where we allow for the optimization and computation of risk-sensitive metrics thanks to the use of distributions over the possible returns Chapter 6. The use of distributions naturally extends to the case when more approximate or varied information is known about the model. Therefore, we adapted uncertain models to take into account distributions over uncertain parameters. We also developed a tractable algorithm exploiting these distributions describing uncertainty in Chapter 7.

In summary, this thesis proposes several novel approaches that address different notions of safe decision making in the face of uncertainty. Furthermore, this thesis makes several significant advances in bridging the gaps between safe policy synthesis methods in MARL and probabilistic model checking. It brings logic and safety games from formal model checking to approximate MARL approaches and brings tractable distributional outlooks to probabilistic model checking. Experimental results demonstrate our methods with both sequential decision making classes have the potential to adapt to different notions of safety and uncertainty.

Safe MARL via Shielding. In Chapter 4, we introduced two novel shielding approaches for safe MARL where each is synthesized based on a Linear Temporal Logic (LTL) safety specification. The factored shielding was designed to improve the scalability

of the centralized method. However, the factored method suffers from unnecessary delays because of the assumption made about shield to shield communication. One avenue of improvement includes extending this work to be completely decentralized thereby increasing applicability in real world scenarios. For example, it is more convenient for drones on a mission to communicate with each other to prevent unsafe situations rather than waiting on local towers which may be hard to implement in certain circumstances.

Logic-guided MARL Reward Shaping. In Chapter 5, we developed a novel method for semi-centralized logic-based MARL reward shaping that is scalable in the number of agents and agnostic to the MARL algorithm. While it is an improvement, as the number of tasks in the logic specification increases, the agents may start to struggle to experience the different reward signals which can confuse the learning process. Moreover, as in Chapter 4, the method is not completely decentralized which means more reliable communication is required between agents. An interesting direction would be to extend this method to the context where the signals between agents are not consistent resulting in additional uncertainty.

Distributional Probabilistic Model Checking. In Chapter 6, we presented a distributional probabilistic model checking approach, which supports a rich set of distributional queries for DTMCs and MDPs. The main advantages are two-fold: we provide a risk-sensitive policy synthesis and more information about any policy to the user. By returning a distribution over the costs for each state, the user can explore the model while computing additional information such as the variance, value-at-risk, probability of a tail event, etc. Additional effort is needed to extend this outlook to other areas of model checking such as partially observable models or multi-objective specifications.

Distributionally Uncertain Probabilistic Model Checking. In Chapter 7, we designed an algorithm to factor in the distribution of the uncertain parameters for parametric uncertain MDP. We represent the distribution using tractable yet expressive distributional representations which can be tuned for better precision. This allows us to better manipulate models where only partial information is available but detailed information about the policy synthesized is important. Furthermore, our approach allows us to take into account both the uncertainty from system modeling errors and that from the unpredictability of real life. While our work focuses on synthesizing a policy for a certain amount of fixed uncertainty, this method can be adapted and included within a learning loop where the uncertainty is gradually refined. This could be helpful for applications that use approximate modeling methods such as model-based RL and used to provide additional guidance to the designer.

8.1 Social and Technological Impact

Achieving the different safety objectives is essential given the AI boom we are currently experiencing. Automated decision making policies are reaching more people and directly affecting more lives. For this reason, it is of paramount importance that we ensure that the risk to users is minimized. Potential applications could include reducing the recommendation of emotionally challenging content when users are experiencing mental struggles. Distributional methods could be used to analyze the medical trajectories of patients, quantify their risks and better inform physicians' decisions. MARL has successfully been applied to client-host network management where certain information may be critical. Shielding or reward shaping techniques previously described could allow for better protection of sensitive data. The various models, methods and the associated safety goals are potentially relevant to smart cities, patient assistance, autonomous robots, network management, search and rescue missions, etc. Finally, the underlying cost structures presented throughout this dissertation can be reformulated for aspects of social responsibility such as algorithm fairness.

Bibliography

[1] A. J. Singh, A. Kumar, and H. C. Lau, "Hierarchical multiagent reinforcement learning for maritime traffic management," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 2020, pp. 1278–1286.

[2] C. Yu, X. Wang, and Z. Feng, "Coordinated multiagent reinforcement learning for teams of mobile sensing robots," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 2019, pp. 2297–2299.

[3] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *arXiv preprint arXiv:1610.03295*, 2016.

[4] P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "A survey and critique of multiagent deep reinforcement learning," *Autonomous Agents and Multi-Agent Systems*, vol. 33, no. 6, pp. 750–797, 2019.

[5] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *arXiv preprint arXiv:1911.10635*, 2019.

[6] M. Kwiatkowska, G. Norman, and J. Sproston, "Probabilistic model checking of the IEEE 802.11 wireless local area network protocol," in *Proc. 2nd Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'02)*, H. Hermanns and R. Segala, Eds., ser. LNCS, vol. 2399, Springer, 2002, pp. 169–187.

[7] M. Kwiatkowska, G. Norman, and J. Sproston, "Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol," *Formal Aspects of Computing*, vol. 14, no. 3, pp. 295–318, 2003.

[8] M. Kwiatkowska, G. Norman, and D. Parker, "The PRISM benchmark suite," in *Proc. 9th International Conference on Quantitative Evaluation of SysTems (QEST'12)*, IEEE CS Press, 2012, pp. 203–204.

[9] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, G. Gopalakrishnan and S. Qadeer, Eds., ser. LNCS, vol. 6806, Springer, 2011, pp. 585–591.

[10] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk, "A storm is coming: A modern probabilistic model checker," in *Proc. 29th International Conference on Computer Aided Verification (CAV'17)*, 2017.

[11] I. ElSayed-Aly, S. Bharadwaj, C. Amato, R. Ehlers, U. Topcu, and L. Feng, "Safe multi-agent reinforcement learning via shielding," in *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, 2021, pp. 483–491.

[12] I. ElSayed-Aly and L. Feng, "Logic-based reward shaping for multi-agent reinforcement learning," *arXiv preprint arXiv:2206.08881*, 2022.

[13] I. Elsayed-Aly, D. Parker, and L. Feng, "Distributional probabilistic model checking," in *NASA Formal Methods Symposium*, Springer, 2024.

[14] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[15] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *AAAI-18: 32nd AAAI Conference on Artificial Intelligence*, 2018, pp. 2669–2678.

[16] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, IEEE, 1977, pp. 46–57.

[17] R. Alur, *Principles of cyber-physical systems*. MIT Press, 2015.

[18] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[19] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE transactions on robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[20] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimality and robustness in multi-robot path planning with temporal logic constraints," *The International Journal of Robotics Research*, vol. 32, no. 8, pp. 889–911, 2013.

[21] M. Hasanbeig, A. Abate, and D. Kroening, "Cautious reinforcement learning with logical constraints," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 2020, pp. 483–491.

[22] A. K. Bozkurt, Y. Wang, M. M. Zavlanos, and M. Pajic, "Control synthesis from linear temporal logic specifications using model-free reinforcement learning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 10 349–10 355.

[23] E. M. Hahn, M. Perez, S. Schewe, F. Somenzi, A. Trivedi, and D. Wojtczak, "Omega-regular objectives in model-free reinforcement learning," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2019, pp. 395–412.

[24] R. Bloem, B. Könighofer, R. Könighofer, and C. Wang, "Shield synthesis," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2015, pp. 533–548.

[25] D. Raju, S. Bharadwaj, and U. Topcu, "Decentralized runtime synthesis of shields for multi-agent systems," *arXiv preprint arXiv:1910.10380*, 2019.

[26] S. Bharadwaj, R. Bloem, R. Dimitrova, B. Konighofer, and U. Topcu, "Synthesis of minimum-cost shields for multi-agent systems," in *2019 American Control Conference (ACC)*, IEEE, 2019, pp. 1048–1055.

[27] Y.-L. Kuo, B. Katz, and A. Barbu, "Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of ltl formulas," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 5604–5610.

[28] M. Elbarbari, K. Efthymiadis, B. Vanderborght, and A. Nowé, "Ltlf-based reward shaping for reinforcement learning," *Adaptive and Learning Agents Workshop*, 2021.

[29] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3387–3395.

[30] X. Li, Z. Serlin, G. Yang, and C. Belta, "A formal methods approach to interpretable reinforcement learning for robotic planning," *Science Robotics*, vol. 4, no. 37, 2019.

[31] M. Guo, J. Tumova, and D. V. Dimarogonas, "Cooperative decentralized multi-agent control under local ltl tasks and connectivity constraints," in *53rd IEEE conference on decision and control*, IEEE, 2014, pp. 75–80.

[32] A. Camacho, R. T. Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith, "Ltl and beyond: Formal languages for reward function specification in reinforcement learning.," in *IJCAI*, vol. 19, 2019, pp. 6065–6073.

[33] C. Neary, Z. Xu, B. Wu, and U. Topcu, "Reward machines for cooperative multi-agent reinforcement learning," *Autonomous Agents and Multi-Agent Systems*, 2021.

[34] A. Nowé, P. Vrancx, and Y.-M. De Hauwere, "Game theory and multi-agent reinforcement learning," in *Reinforcement Learning*, Springer, 2012, pp. 441–470.

[35] L. Buşoniu, R. Babuška, and B. De Schutter, "Multi-agent reinforcement learning: An overview," *Innovations in multi-agent systems and applications-1*, pp. 183–221, 2010.

[36] C. J. C. H. Watkins, "Learning from delayed rewards," *King's College, Cambridge*, 1989.

[37] K. Tuyls, P. J. Hoen, and B. Vanschoenwinkel, "An evolutionary dynamical analysis of multi-agent learning in iterated games," *Autonomous Agents and Multi-Agent Systems*, vol. 12, no. 1, pp. 115–153, 2006.

[38] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in neural information processing systems*, 2017, pp. 6379–6390.

[39] M. Ummels and C. Baier, "Computing quantiles in Markov reward models," in *Proc. 16th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'13)*, ser. LNCS, vol. 7794, Springer, 2013, pp. 353–368.

[40] M. Randour, J.-F. Raskin, and O. Sankur, "Percentile queries in multi-dimensional Markov decision processes," *Formal methods in system design*, vol. 50, pp. 207–248, 2017.

[41] J. Klein, C. Baier, P. Chrszon, *et al.*, "Advances in probabilistic model checking with PRISM: Variable reordering, quantiles and weak deterministic Büchi automata," *International Journal on Software Tools for Technology Transfer*, vol. 20, no. 2, pp. 179–194, 2018.

[42] A. Hartmanns, S. Junges, J.-P. Katoen, and T. Quatmann, "Multi-cost bounded tradeoff analysis in MDP," *Journal of Automated Reasoning*, vol. 64, no. 7, pp. 1483–1522, 2020.

[43] M. Chen, J. Katoen, L. Klinkenberg, and T. Winkler, "Does a program yield the right distribution? - verifying probabilistic programs via generating functions," in *Proc. 34th International Conference on Computer Aided Verification (CAV'22)*, ser. LNCS, vol. 13371, Springer, 2022, pp. 79–101.

[44] T. Brazdil, K. Chatterjee, V. Forejt, and A. Kucera, "Trading performance for stability in Markov decision processes," *Journal of Computer and System Sciences*, vol. 84, pp. 144–170, 2017.

[45] M. J. Sobel, "The variance of discounted Markov decision processes," *Journal of Applied Probability*, vol. 19, no. 4, pp. 794–802, 1982.

[46] J. A. Filar, D. Krass, and K. W. Ross, "Percentile performance criteria for limiting average Markov decision processes," *IEEE Transactions on Automatic Control*, vol. 40, no. 1, pp. 2–10, 1995.

[47] A. Majumdar and M. Pavone, "How should a robot assess risk? towards an axiomatic theory of risk in robotics," in *Robotics Research: The 18th International Symposium ISRR*, Springer, 2020, pp. 75–84.

[48] J. Křetínskỳ and T. Meggendorfer, "Conditional value-at-risk for reachability and mean payoff in Markov decision processes," in *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, 2018, pp. 609–618.

[49] Y. Chow, A. Tamar, S. Mannor, and M. Pavone, "Risk-sensitive and robust decision-making: A cvar optimization approach," *Advances in neural information processing systems*, vol. 28, 2015.

[50] T. Meggendorfer, "Risk-aware stochastic shortest path," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, 2022, pp. 9858–9867.

[51] M. Rigter, P. Duckworth, B. Lacerda, and N. Hawes, "Planning for risk-aversion and expected value in MDPs," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 32, 2022, pp. 307–315.

[52] V. Borkar and R. Jain, "Risk-constrained Markov decision processes," *IEEE Transactions on Automatic Control*, vol. 59, no. 9, pp. 2574–2579, 2014.

[53] Y. Chow and M. Ghavamzadeh, "Algorithms for CVaR optimization in MDPs," *Advances in neural information processing systems*, vol. 27, 2014.

[54] M. Cubuktepe and U. Topcu, "Verification of markov decision processes with risk-sensitive measures," in *Proc. Annual American Control Conference (ACC'18)*, IEEE, 2018, pp. 2371–2377.

[55] S. Jha, V. Raman, D. Sadigh, and S. A. Seshia, "Safe autonomy under perception uncertainty using chance-constrained temporal logic," *Journal of Automated Reasoning*, vol. 60, no. 1, pp. 43–62, 2018.

[56] M. G. Bellemare, W. Dabney, and M. Rowland, *Distributional Reinforcement Learning*. MIT Press, 2023, http://www.distributional-rl.org.

[57] C. Lyle, M. G. Bellemare, and P. S. Castro, "A comparative analysis of expected and distributional reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4504–4511.

[58] N. Bäuerle and J. Ott, "Markov decision processes with average-value-at-risk criteria," *Mathematical Methods of Operations Research*, vol. 74, no. 3, pp. 361–379, 2011.

[59] S. C. Hora, "Aleatory and epistemic uncertainty in probability elicitation with an example from hazardous waste management," *Reliability Engineering & System Safety*, vol. 54, no. 2-3, pp. 217–223, 1996.

[60] E. Hüllermeier and W. Waegeman, "Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods," *Machine Learning*, vol. 110, pp. 457–506, 2021.

[61] X. Pan, D. Seita, Y. Gao, and J. Canny, "Risk averse robust adversarial reinforcement learning," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 8522–8528.

[62] A. Guez, D. Silver, and P. Dayan, "Efficient bayes-adaptive reinforcement learning using sample-based search," *Advances in neural information processing systems*, vol. 25, 2012.

[63] M. Rigter, B. Lacerda, and N. Hawes, "Risk-averse bayes-adaptive reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 1142–1154, 2021.

[64] J. Queeney and M. Benosman, "Risk-averse model uncertainty for distributionally robust safe reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[65] H. Xu and S. Mannor, "Distributionally robust markov decision processes," *Advances in Neural Information Processing Systems*, vol. 23, 2010.

[66] A. Nilim and L. El Ghaoui, "Robust control of markov decision processes with uncertain transition matrices," *Operations Research*, vol. 53, no. 5, pp. 780–798, 2005.

[67] E. Delage and Y. Ye, "Distributionally robust optimization under moment uncertainty with application to data-driven problems," *Operations research*, vol. 58, no. 3, pp. 595–612, 2010.

[68] D. Kuhn, P. M. Esfahani, V. A. Nguyen, and S. Shafieezadeh-Abadeh, "Wasserstein distributionally robust optimization: Theory and applications in machine learning," in *Operations research & management science in the age of analytics*, Informs, 2019, pp. 130–166.

[69] Z. Hu and L. J. Hong, "Kullback-leibler divergence constrained distributionally robust optimization," *Available at Optimization Online*, vol. 1, no. 2, p. 9, 2013.

[70] M. Petrik and R. H. Russel, "Beyond confidence regions: Tight bayesian ambiguity sets for robust mdps," *Advances in neural information processing systems*, vol. 32, 2019.

[71] Z. Chen, P. Yu, and W. B. Haskell, "Distributionally robust optimization for sequential decision-making," *Optimization*, vol. 68, no. 12, pp. 2397–2426, 2019.

[72] Y. Lin, Y. Ren, and E. Zhou, "Bayesian risk markov decision processes," *Advances in Neural Information Processing Systems*, vol. 35, pp. 17 430–17 442, 2022.

[73] S. Junges, E. Ábrahám, C. Hensel, *et al.*, "Parameter synthesis for markov models," *arXiv preprint arXiv:1903.07993*, 2019.

[74] T. Badings, M. Cubuktepe, N. Jansen, S. Junges, J.-P. Katoen, and U. Topcu, "Scenario-based verification of uncertain parametric mdps," *International Journal on Software Tools for Technology Transfer*, vol. 24, no. 5, pp. 803–819, 2022.

[75] M. Cubuktepe, N. Jansen, S. Junges, J.-P. Katoen, and U. Topcu, "Scenario-based verification of uncertain mdps," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2020, pp. 287–305.

[76] T. Badings, S. Junges, A. Marandi, U. Topcu, and N. Jansen, "Efficient sensitivity analysis for parametric robust markov chains," *arXiv preprint arXiv:2305.01473*, 2023.

[77] B. Salmani and J.-P. Katoen, "Automatically finding the right probabilities in bayesian networks," *Journal of Artificial Intelligence Research*, vol. 77, pp. 1637–1696, 2023.

[78] L. Bortolussi, F. Cairoli, G. Carbone, and P. Pulcini, "Scalable stochastic parametric verification with stochastic variational smoothed model checking," in *International Conference on Runtime Verification*, Springer, 2023, pp. 45–65.

[79] M. Suilen, T. D. Simão, D. Parker, and N. Jansen, "Robust anytime learning of markov decision processes," *Advances in Neural Information Processing Systems*, vol. 35, pp. 28 790–28 802, 2022.

[80] E. M. Wolff, U. Topcu, and R. M. Murray, "Robust control of uncertain markov decision processes with temporal logic specifications," in *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, IEEE, 2012, pp. 3372–3379.

[81] E. Bacci and D. Parker, "Probabilistic guarantees for safe deep reinforcement learning," in *Formal Modeling and Analysis of Timed Systems: 18th International Conference, FORMATS 2020, Vienna, Austria, September 1–3, 2020, Proceedings 18*, Springer, 2020, pp. 231–248.

[82] E. Bacci and D. Parker, "Verified probabilistic policies for deep reinforcement learning," in *NASA Formal Methods Symposium*, Springer, 2022, pp. 193–212.

[83] T. Badings, L. Romao, A. Abate, and N. Jansen, "Probabilities are not enough: Formal controller synthesis for stochastic dynamical models with epistemic uncertainty," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 14 701–14 710.

[84] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330–337.

[85] Y.-M. De Hauwere, P. Vrancx, and A. Nowé, "Learning multi-agent state space representations," in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, 2010, pp. 715–722.

[86] J. Kemeny, J. Snell, and A. Knapp, *Denumerable Markov Chains*, 2nd. Springer-Verlag, 1976.

[87] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," *Formal Methods in System Design*, vol. 19, no. 3, pp. 291–314, 2001.

[88] A. Pnueli, "The temporal semantics of concurrent programs," *Theoretical Computer Science*, vol. 13, pp. 45–60, 1981.

[89] S. Li, L. Da Xu, and S. Zhao, "The internet of things: A survey," *Information Systems Frontiers*, vol. 17, no. 2, pp. 243–259, 2015.

[90] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," *Formal Methods in System Design*, vol. 19, no. 3, pp. 291–314, 2001.

[91] F. S. Melo and M. Veloso, "Learning of coordination: Exploiting sparse interactions in multiagent systems," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, 2009, pp. 773–780.

[92] J. Yang, A. Nakhaei, D. Isele, K. Fujimura, and H. Zha, "Cm3: Cooperative multi-goal multi-stage multi-agent reinforcement learning," in *International Conference on Learning Representations*, 2019.

[93] R. Ehlers and V. Raman, "Slugs: Extensible gr (1) synthesis," in *International Conference on Computer Aided Verification*, Springer, 2016, pp. 333–339.

[94] S. Carr, N. Jansen, S. Junges, and U. Topcu, "Safe reinforcement learning via shielding under partial observability," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 14 748–14 756.

[95] L. Feng, M. Kwiatkowska, and D. Parker, "Automated learning of probabilistic assumptions for compositional reasoning," in *International Conference on Fundamental Approaches to Software Engineering*, Springer, 2011, pp. 2–17.

[96] J. Křetínskỳ, T. Meggendorfer, and S. Sickert, "Owl: A library for ω-words, automata, and ltl," in *International Symposium on Automated Technology for Verification and Analysis*, Springer, 2018, pp. 543–550.

[97] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, E. Renault, and L. Xu, "Spot 2.0 — a framework for LTL and ω-automata manipulation," in *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*, ser. Lecture Notes in Computer Science, vol. 9938, Springer, Oct. 2016, pp. 122–129. DOI: 10.1007/978-3-319-46520-3_8.

[98] F. Christianos, L. Schäfer, and S. V. Albrecht, "Shared experience actor-critic for multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems*, 2020.

[99] G. Papoudakis, F. Christianos, L. Schäfer, and S. V. Albrecht, "Comparative evaluation of multi-agent deep reinforcement learning algorithms," *arXiv preprint arXiv:2006.07869*, 2020.

[100] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal logic-based reactive mission and motion planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[101] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.

[102] A. Bianco and L. de Alfaro, "Model checking of probabilistic and nondeterministic systems," in *Proc. 15th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'95)*, ser. LNCS, vol. 1026, Springer, 1995, pp. 499–513.

[103] S. Haddad and B. Monmege, "Reachability in MDPs: Refining convergence of value iteration," in *Proc. 8th International Workshop on Reachability Problems (RP'14)*, ser. LNCS, vol. 8762, Springer, 2014, pp. 125–137.

[104] R. T. Rockafellar and S. Uryasev, "Conditional value-at-risk for general loss distributions," *Journal of banking & finance*, vol. 26, no. 7, pp. 1443–1471, 2002.

[105] L. Feng, C. Wiltsche, L. Humphrey, and U. Topcu, "Synthesis of human-in-the-loop control protocols for autonomous systems," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 450–462, 2016.

[106] W. Dabney, G. Ostrovski, D. Silver, and R. Munos, "Implicit quantile networks for distributional reinforcement learning," in *International conference on machine learning*, PMLR, 2018, pp. 1096–1105.

[107] Z. Liu, Q. Bai, J. Blanchet, *et al.*, "Distributionally robust Q-learning," in *International Conference on Machine Learning*, PMLR, 2022, pp. 13 623–13 643.