Constrained Exploration in Reinforcement Learning with Optimality Preservation

Peter C. Y. Chen
Department of Mechanical Engineering
National University of Singapore
Singapore 117576
Email: mpechenp@nus.edu.sg

Abstract

We consider a class of reinforcement-learning systems in which the agent follows a behavior policy to explore a discrete state-action space to find an optimal policy while adhering to some restriction on its behavior. Such restriction may prevent the agent from visiting some state-action pairs, possibly leading to the agent finding only a sub-optimal policy. To address this problem we introduce the concept of constrained exploration with optimality preservation, whereby the exploration behavior of the agent is constrained to meet a specification while the optimality of the (original) unconstrained learning process is preserved. We first establish a feedback-control structure that models the dynamics of the unconstrained learning process. We then extend this structure by adding a supervisor to ensure that the behavior of the agent meets the specification, and establish (for a class of reinforcement-learning problems with a known deterministic environment) a necessary and sufficient condition under which optimality is preserved. This work demonstrates the utility and the prospect of studying reinforcement-learning problems in the context of the theories of discrete-event systems, automata and formal languages.

Keywords Constrained exploration, desired behavior, supervision, optimality preservation, automata

1 Introduction

In reinforcement learning, exploration refers to the agent taking actions according to a behavior policy in order to traverse a typically discrete state space and collect rewards. While exploring the state space, the agent uses an update rule to estimate, based on the rewards collected, the Q-values (i.e., state-action values) from one iteration to the next. If the Q-values converge to their optimums, an optimal policy can then be obtained. For a class of reinforcement learning problems, such convergence is guaranteed under the Robbins-Monro conditions [47].

A requirement for satisfying the Robbins-Monro conditions is that every state-action pair must have a non-zero probability of being visited by the agent — also known as persistent exploration. If we consider the agent taking an action (when it is at a state) as 'generating' a symbol denoting that action, the sequences of actions thus generated by the agent as it traverses through the states represent the behavior of the agent. For an episodic learning process, the behavior of the agent consists of all possible action sequences from the initial state to the set of goal states. We refer to such a process as an unconstrained learning process, and the associated optimal Q-values as the intrinsic optimums.

1.1 The constrained exploration problem

It may be useful to require the agent to exhibit certain desired behavior during exploration. The typical motivation for imposing such requirements is to achieve better performance in a learning process. For example, we may require the agent to avoid some states that may be considered unsafe, or some action sequences that are considered undesirable (such as the case of a robot repeatedly touching the ball in robot soccer as described in [42]).

In practice, we might also need to impose certain requirements just to accommodate some operational characteristics of the agent itself. For instance, suppose that due to some technical glitch in its motor drive, a wheeled robot (i.e., the agent) is more likely to malfunction if it is commanded to take two or more consecutive right turns. To reduce the risk of robot failure, we might impose the requirement that during exploration the robot is not allowed to take two right turns in succession. Imposition of this type of requirements may not necessarily be motivated by the goal of achieving better learning performance; it could simply be a way to cope with certain (possibly unexpected) limitation of the operational capability of the agent.

To meet these requirements necessitates a mechanism to manipulate the behavior of the agent during the learning process. We refer to such a mechanism as a supervisor. We consider a supervisor as effective if it forces the agent to stay within a prescribed subset of its unconstrained behavior. We call the learning process in which the agent is under such manipulation a constrained learning process.

Since the behavior of the agent is described by the sequences of actions taken by the agent as it traverses through the states, manipulating the behavior of the agent means that the agent may be prevented by the supervisor from taking certain actions when it reaches certain states. Such manipulation raises the possibility that some state-action pairs will not be visited by the agent throughout the entire learning process, thus breaking persistent exploration. As a result, the Q-values may not converge to their intrinsic optimums. The following example illustrates this issue.

Example 1.1. Figure 1 illustrates two cases of desired behavior of an agent traversing in a 4×4 grid environment. In the first case, illustrated in Figure 1(a), the agent is not allowed to take two consecutive right (i.e., a_2) actions , while in the second, Figure 1(b), the agent is allowed to take a right action only immediately after having taken two consecutive up (i.e., a_1) actions.

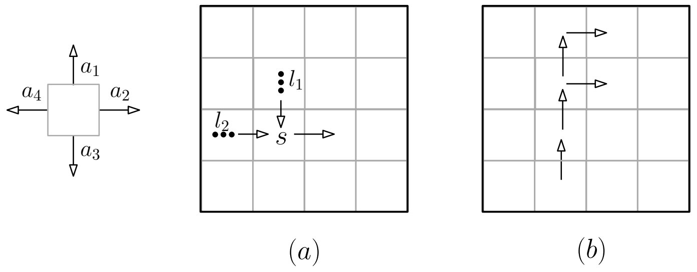

Figure 1: Left: The actions at a state. (a) and (b): Two cases of desired behavior.

Consider the situation in Figure 1(a). If the agent reaches s by following the action sequence l_2, it is then prohibited from taking the right action at s; however, it is permitted to do so if it reaches s via l_1. Hence, all of the state-action pairs (s, a_i), where $i \in \{1, 2, 3, 4\}$, have an non-zero probability of being visited by the agent. For the situation in Figure 1(b), the agent is allowed to take a right action only immediately after having taken two consecutive up (i.e., a_1) actions. This results in the agent being prevented from visiting all state-action pairs (s, a_2), where s is any state in the lower two rows of the grid; hence, the probability of these state-action pairs being visited by the agent is zero.

Ideally we seek a supervisor that, while manipulating the exploration behavior of the agent, does not interfere with the Q-values converging to their intrinsic optimums. We refer to such a supervisor as optimality-preserving.

We describe the constrained exploration problem as follows. For a given desired behavior of the agent, find a supervisor that is both effective and optimality-preserving.

In this paper we investigate the constrained exploration problem by first developing an automaton-model that describes the dynamics of the unconstrained reinforcement learning process, then formulating a supervisory feedback-control structure incorporating this automaton-model. We next prove (for a given desired behavior of the agent) the existence of an effective supervisor and establish its realization in the form of an automaton without any knowledge about the dynamics of the environment. We then determine a necessary and sufficient condition under which an effective supervisor is optimality-preserving for a class of reinforcement-learning problems under the assumption that the dynamics of the environment is deterministic and known.

1.2 Literature survey

In the standard setting of a reinforcement learning problem, the exploration behavior of the agent is determined by the behavior policy (e.g., ϵ-greedy) that governs the action-selection process. By incorporating external knowledge into the action-selection process, the exploration behavior of the agent can be manipulated to improve the speed of convergence or to meet certain operational requirements.

1.2.1 Incorporating external knowledge for faster convergence

Incorporating external knowledge for the purpose of improving the speed of convergence enables policies to be learned based on additional information about the structure of the problem or based on expert intervention. Techniques that are based on additional information about the structure of the problem include reward shaping by modification of the reward function (e.g., [42, 33]), behavior cloning (e.g., [41]), direct policy learning (e.g., [38, 49, 55]), and inverse reinforcement learning (e.g., [1]). Techniques for improving the speed of learning based on expert intervention include teacher-student framework (e.g., [53, 4]), human-feedback for policy shaping (e.g., [12, 24]), transfer of learned actions or skill priors (e.g.,[50, 45]), and human-in-the-loop action removal or selection (e.g., [2, 48, 37]).

1.2.2 Incorporating external knowledge for meeting requirements

Incorporating external knowledge in order to manipulate exploration behavior also plays a fundamental role in solving problems in which the speed of convergence is of secondary importance to operational requirements. One line of research involves devising some mechanism (reward-related or otherwise) to force the agent to conform to certain operational specifications that are expressed in temporal logic [5, 36, 9, 10, 35, 29, 25, 23]. Another is to cast the problems in the context of safe reinforcement learning, where the objective is to ensure that the agent achieves some acceptable level of return while meeting safety requirements [17, 44]. Safety in this context is usually quantified by some risk measure or is characterized by some prescribed behavior of the agent. Risk-based approaches for safe reinforcement learning use risk measures that are mostly derived from (i) the variance of return and (ii) the probability of the agent entering an error state or visiting some undesirable state-action pair. For such approaches, external knowledge is incorporated for risk evaluation, while meeting safety requirements means minimizing the risk.

Two current main trends of risk-averse reinforcement learning are (1) to modify the optimization criteria so as to include some form of risk measure, and (2) to use a risk measure directly for action selection. Techniques of the first trend include those that (i) use a worst-case criterion, where a policy is considered to be optimal if it has the maximum worst-case return [27, 19, 16], (ii) use risk-sensitive criteria based on exponential functions [39, 21], and (iii) introduce safety-related constraints into the problem formulation so that solutions can be obtained under the framework of Constrained Markov Decision Processes [22, 40]. The second trend focuses on retaining the original (i.e., non-safety related) optimization criteria while using a risk measure to directly influence action selection. This leads to a variety of techniques, including those that (i) restrict exploration to a set of safe policies [13] or a set of states that are considered controllable [20], (ii) regulate the exploration via safety signals [14], and (iii) ensure that exploration is stable in the sense of Lyapunov (e.g., [7]).

Safety can also be characterized by certain prescribed behavior of the agent, with the prescription expressed in symbolic logic. This is the formalism in the approach of shielding, where external knowledge is encapsulated in logic-based specifications that prescribe the safe behavior, and the agent is manipulated to meet these specifications during exploration [3, 52, 30, 26, 6, 15].

1.3 Novelty, key limitation, and application context

Whether for improving speed of convergence or for meeting requirements on exploration behavior, essentially all the approaches cited in Section 1.2 aim to guide the agent in avoiding certain states or state-action pairs that are considered undesirable or unsafe, under the premise that visiting them may result in a deterioration of expected return, violation of some requirement, or at the extreme a catastrophic outcome. Since avoiding a specific state implies avoiding some action(s) at the predecessor of the specified state, state avoidance is subsumed under state-action avoidance, which is generally referred to as action pruning in the reinforcement learning literature. In these approaches, action pruning is static in the sense that, if an action is to be avoided by the agent at a state, it will be avoided at that state for all visits during the entire learning process. Such static action pruning breaks persistent exploration, giving rise to the possibility of finding only policies that are sub-optimal (i.e., when compared to an optimal policy obtainable without any safety considerations).

The problem of constrained exploration (with optimality preservation) as proposed in this paper is also about manipulating the exploration behavior of the agent. The key technical difference between our proposed approach and those reported in the literature (and thereby the novelty) is that in our approach action pruning is dynamic in the sense that the decision of whether to prevent the agent from taking a specific action at a state depends not on the state itself but on how that state is reached. In other words, this decision depends on the sequence of actions that the agent has executed in order to reach that state. With dynamic action pruning, the agent may be prevented by the supervisor from taking a specific action on one visit to a state but be permitted to take that action on another visit to the same state, if these two visits are resulted from two different sequences of actions (both starting from the initial state). Such supervision admits the possibility that all state-action pairs are 'visitable', thus enabling the agent to maintain persistent exploration even though its behavior is constrained by some specification throughout the learning process. In particular, the construction of such a supervisor is independent of the dynamics of the environment.

Moreover, the establishment in this work of a necessary and sufficient condition (on the specification) for maintaining persistent exploration makes it possible to determine a priori, for the case where the environment dynamics is assumed to be deterministic and known, whether optimality with respect to the unconstrained problem can still be achieved under constrained exploration. Although there exist approaches (such as Constrained Markov Decision Process discussed in Section 1.2) for solving a constrained reinforcement-learning problem under the same assumption, these approaches do no directly address the question of whether optimality is preserved without actually solving the problem.

While the implementation of constrained exploration (under a supervisor that performs dynamic action pruning) does not require any knowledge about the environment, the method proposed in this work for determining whether optimality is preserved in such a setting does require the dynamics of the environment to be deterministic and known. This requirement represents a key limitation on the applicability of the theoretical results reported in this paper, which is elaborated further in Section 6.

Despite this limitation, the proposed approach remains useful in certain application domains of reinforcement learning. One such domain is the Teacher-Student framework e.g., [53, 56]. In this framework, the teacher is an expert who provides guidance to a student (i.e., the learning agent). The teacher typically has access to a complete model of the environment. Under this framework, the teacher may attempt to guide the student by restricting the actions that the student is allowed to take at a given state. With a deterministic environment, the consequence of an action taken by the student is predictable, which enables the teacher to craft more effective guidance, e.g., for improving the speed of learning or ensuring the safety of the student. In addition, when establishing objectives for teaching or for the student's learning, it would be desirable that (before the student commences the learning process) the teacher is able to determine whether optimality with respect to the unconstrained setting can still be achieved while the learning behavior of the student is constrained by some given specification. With the assumption that the dynamics of the environment is deterministic, our proposed approach can be directly applied to facilitate this determination.

This application context (in which learning is conducted in a known deterministic environment) can be extended to the setting that integrates Curriculum Learning into the Teacher-Student framework. Under such a framework, the teacher can design a curriculum that gradually introduces the student to more complex tasks. For instance, a curriculum can include learning tasks that transition from solving deterministic problems to non-deterministic ones. This allows the student to focus on basic tasks at the early stage of learning without having to deal with the added complexity of uncertainty and randomness, thus helping the student learn more effectively.

1.4 Related work

In developing our proposed approach, we employ a general methodology for studying two interacting state-transition systems. In our problem setting, the specification is represented by the language generated by a state-transition model (in the form of an automaton), and the dynamics of the agent when unconstrained is modeled by another a state-transition model (again, an automaton). The exploration behavior of the agent that meets this specification is then investigated in the context of the ‘synchronized’ operation of these two state-transition models.

This general methodology is also employed in a number of existing approaches on the subject of manipulating the exploration behavior of the agent to satisfy given specifications. In these approaches, the specifications are first expressed in temporal logic then converted into some state-transition model, and the state-transition models representing the specifications and the dynamics of agent-environment interaction are in the form of either automata or Markov Decision Processes. Although the specific objectives of these existing approaches are different from that presented in this paper, in terms of methodology these existing approaches collectively represent the state-of-the-art to which our work is the most closely related. Littman et al. [36] and Brafman et al. [9] studied the specficication of non-Markovian rewards using linear temporal logic; De Giacomo et al. proposed the concept of restraining bolts to force the agent to exhibit certain prescribed behavior [23]; Alshiekh et al. [3] and den Hengst et al. [15] proposed the technique of shielded learning, based on the concept of shielding in reactive systems [31], to enforce safety specifications; Hasanbeig et al. proposed a method for limiting the extent of exploration to a portion that is relevant to satisfying some logic-based constraints so as to improve the speed of learning [25] and developed the concept of safe padding to guide the agent in conforming to safety specifications [26]. In these approaches, an optimal policy is sought for the constrained reinforcement-learning problem resulted from the imposition of specifications on the behavior of the agent, while in our proposed approach we still seek an optimal policy for the unconstrained case (i.e., to preserve optimality) even though the behavior of the agent is also constrained to meet certain specifications. Such optimality preservation is made possible by the dynamic action-pruning mechanism of the supervisor that enables the agent to maintain persistent exploration while meeting the specifications.

1.5 Organization

Section 2 summarizes notations and terminologies related to the existing concepts and techniques that are used in the subsequent analysis. Section 3 presents the proposed automaton-model that describes the dynamics of the class of unconstrained reinforcement-learning systems investigated in this paper. Section 4 presents the structure and characteristics of supervision, and discusses automaton representation of desired behavior. Section 5 proves the existence of an effective supervisor for a given desired behavior, and presents a necessary and sufficient condition for preserving optimality in a reinforcement-learning process under supervision. Section 6 discusses the implications, limitations, and possible extension of our approach. Section 7 presents the conclusions.

2 Preliminaries

The set of real numbers is denoted by $\mathbb{R}$, and the set of non-negative integers by $\mathbb{N}_0$. A closed interval between $a \in \mathbb{R}$ and $b \in \mathbb{R}$ is denoted by $[a, b]$, and an integer interval between $i \in \mathbb{N}_0$ and $j \in \mathbb{N}_0$ by $[\![i, j]\!]$. Let $|B|$ denote the number of elements in a finite set B, and 2^B denote the power set of B. Let $\varnothing$ denote the empty set. An alphabet is a nonempty finite set (denoted by $\mathcal{A}$) whose member are symbols. A string over $\mathcal{A}$ is a sequence of symbols each belonging to $\mathcal{A}$. In this paper, we use the terms string and action sequence interchangeably depending on the context. The concatenation of two strings u and v, written as uv, is the string consisting of u followed by v. Let $|s|$ denote the length of a string s, i.e., the number of symbols in s. Let λ denote the empty string with the properties: (i) $|\lambda| = 0$, and (ii) $\lambda\, l = l\, \lambda = l$ for any string l. Let $\mathcal{A}^+$ be the set of all possible strings over $\mathcal{A}$. Let $\mathcal{A}^* = \lambda \cup \mathcal{A}^+$. If $l = usv$ with $u, s, v \in \mathcal{A}^*$, then we call u a prefix of l (written as $u \leq l$) and s a substring of l (written as $s \prec l$). A language over $\mathcal{A}$ is a subset of $\mathcal{A}^*$. The prefix-closure of a language L, denoted by $\overline{L}$, is defined as: $\overline{L} = \{u \in \mathcal{A}^* \,|u \leq l \text{ for some } l \in L\}$. A language L is prefix-closed if $L = \overline{L}$. (In this paper, we use L as a generic symbol to denote a language and use the calligraphic form $\mathcal{L}$ to denote a 'specification language'.)

A deterministic automaton consists of six elements expressed as:

$$(\mathcal{S}, \mathcal{A}, \delta, \Gamma, s^\circ, \mathcal{S}^\bullet)$$

where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of events (i.e., an alphabet); $\delta : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is called the transition function; $\Gamma : \mathcal{S} \to 2^{\mathcal{A}}$ is called the active event function; $s^\circ \in \mathcal{S}$ is the initial state; and $\mathcal{S}^\bullet \in \mathcal{S}$ is the set of marked states. We say that a transition $\delta(s,a)$ is defined, written as $\delta(s,a)!$, if there is a state $s' \in \mathcal{S}$ such that $\delta(s,a) = s'$. The members of $\mathcal{A}$ are called actions in reinforcement learning literature; in this paper, we will follow this convention. For a state s, $\Gamma(s)$ specifies the set of all actions a for which $\delta(s,a)$ is defined. We refer to $\Gamma(s)$ as the active event set at s. In a typical reinforcement-learning problem, $\Gamma(s)$ contains all the actions available for selection by the agent at state s, i.e., $\Gamma(s) = \mathcal{A}$.

The transition function δ can be extended from $\mathcal{S}\times\mathcal{A}$ to $\mathcal{S}\times\mathcal{A}^*$ according to the following rules: $\delta(s,\lambda) = s$ and $\delta(s, la) = \delta(\delta(s,l),a)$, where $s \in \mathcal{S}$, $l \in \mathcal{A}^*$, and $a \in \mathcal{A}$, provided that $\delta(s,l)$ and $\delta(\delta(s,l),a)$ are defined. A state $s \in \mathcal{S}$ is reachable (coreachable) if there is a string $l \in \mathcal{A}^*$ such that $\delta(s^\circ, l) = s$ (resp. $\delta(s,l) \in \mathcal{S}^\bullet$). An automaton is trim if all of its states are both reachable and coreachable.

We use a bold capital letter to represent a specific automaton, in which case each element of the automaton is assigned a subscript in the form of the lower case of that letter. For instance, $\mathrm{X} = (\mathcal{S}_x, \mathcal{A}_x, \delta_x, \Gamma_x, s_x^\circ, \mathcal{S}_x^\bullet)$. Let $\mathcal{R}_e[\mathrm{X}]$ denotes the reachable part of X. The language generated by X, denoted by $L(\mathrm{X})$, is defined as: $L(\mathrm{X}) := \{l \in \mathcal{A}_x^* \,|\, \delta_x(s_x^\circ, l)!\}$. The language marked (or recognized) by X, denoted by $L_m(\mathrm{X})$, is defined as: $L_m(\mathrm{X}) := \{l \in \mathcal{A}_x^* \,|\, \delta_x(s_x^\circ, l) \in \mathcal{S}_x^\bullet\}$. If X is trim, then $L(\mathrm{X}) = \overline{L_m(\mathrm{X})}$. A language is regular if it is recognized by some finite automaton. We say that two strings $u, v \in \mathcal{A}_x^*$ are equivalent with respect to X, written as $u \sim_\mathrm{X} v$, if there exists a state $s \in \mathcal{S}_x$ such that $\delta(s_x^\circ, u) = \delta(s_x^\circ, v) = s$. The relation $\sim_\mathrm{X}$ divides $\mathcal{A}_x^*$ into equivalence classes, one for each state that is reachable from s_x° [28]. We denote the equivalent class corresponding to state s as $\mathbb{C}_x(s)$.

The product of two automata X and Y is defined as [11]

$$\mathrm{Z} := \mathrm{X}\|\,\mathrm{Y} = \mathcal{R}_e\big[(\mathcal{S}_z, \mathcal{A}_z, \delta_z, \Gamma_z, (s_x^\circ, s_y^\circ), \mathcal{S}_z^\bullet)\big]$$

where $\mathcal{S}_z = \mathcal{S}_x \times \mathcal{S}_y$, $\mathcal{A}_z = \mathcal{A}_x \cup \mathcal{A}_y$, $\mathcal{S}_z^\bullet = \mathcal{S}_x^\bullet \times \mathcal{S}_y^\bullet$, $\Gamma_z((s_x, s_y)) = \Gamma_x(s_x) \cap \Gamma_y(s_y)$ with $(s_x, s_y) \in \mathcal{S}_z$, and the joint transition function δ_z is defined as

$$\delta_z((s_x, s_y),\, a) = \begin{cases} \big(\delta_x(s_x,\, a),\, \delta_y(s_y,\, a)\big) & \text{if } a \in \Gamma_x(s_x) \cap \Gamma_y(s_y) \\ \text{undefined} & \text{otherwise} \end{cases}$$

It follows from the definition of δ_z that $l \in \mathbb{C}_z((s_x, s_y))$ implies that $l \in \mathbb{C}_y(s_y) \cap \mathbb{C}_x(s_x)$. If $\mathcal{A}_x \subseteq \mathcal{A}_y$ and $L(\mathrm{X}) \subseteq L(\mathrm{Y})$, then X is isomorphic to $\mathrm{X}\|\mathrm{Y}$ with state renaming. Hence, we have $L(\mathrm{X}\|\mathrm{Y}) = L(\mathrm{X})$ and $L_m(\mathrm{X}\|\mathrm{Y}) = L_m(\mathrm{X})$.

We next introduce the notion of the probabilistic language generated by a stochastic system consisting of an automation with a state-wise action distribution. Detailed definition and analysis concerning such a system have been presented in [34, 43, 18]. We highlight here the key points relevant to analysis presented in this paper. Consider an automaton X. At each state $s \in \mathcal{S}_x$, assign a discrete probability distribution π_b over $\mathcal{A}_x$ such that $\pi_b(s,a) > 0$ if and only if $\delta_x(s,a)!$ for all $a \in \mathcal{A}_x$ and $\sum_{a \in \mathcal{A}_x} \pi_b(s,a) = 1$. The probabilistic language generated by (X, π_b), denoted by $L^p_{\mathrm{X}}(l)$, is defined as: $L^p_{\mathrm{X}} : \mathcal{A}_x^* \to [0,1]$, where $L^p_{\mathrm{X}}(\lambda) = 1$, and for all $l \in L(\mathrm{X})$ and $a \in \mathcal{A}_x$,

$$L^p_{\mathrm{X}}(la) = \begin{cases} L^p_{\mathrm{X}}(l)\,\pi_b\left(\delta_x(s_x^\circ, l), a\right) & \text{if } \delta_x(s_x^\circ, l)! \\ L^p_{\mathrm{X}}(la) = 0 & \text{otherwise} \end{cases}$$

Informally, $L^p_{\mathrm{X}}(l)$ is the probability of the string l being generated by X.

3 Automaton Model of Reinforcement Learning

Reinforcement-learning problems are typically formulated as Markov Decision Processes, in which the dynamics of the interaction between the agent and its environment is described by a state-transition structure. If we view the agent taking an action (when it is at a state) as 'generating' a symbol denoting that action, then the sequences of actions thus generated by the agent (as it traverses through the states) can be considered to form a formal language that describes the exploration behavior of the agent. This perspective motivates the choice of formulating reinforcement-learning problems in the framework of automata and formal language theory, with the benefit that this framework offers a rich set of concepts and techniques for studying such dynamical behavior.

In this section we formulate the dynamics of an reinforcement-learning process with unconstrained exploration in the framework of automata and formal language theory.

3.1 Unconstrained learning behavior

We consider the class of episodic reinforcement-learning problems where an agent operates in an environment modeled by the trim automaton:

$$\mathrm{G} = (\mathcal{S}_g, \mathcal{A}_g, \delta_g, \Gamma_g, s_g^\circ, \mathcal{S}_g^\bullet) \tag{1}$$

with the following properties: (i) $|\mathcal{S}_g| < \infty$; (ii) the transition function δ_g is total, i.e., $\delta_g(s,a)!$ for all $(s,a) \in \mathcal{S}_g \times \mathcal{A}_g$; and (iii) s_g° and $\mathcal{S}_g^\bullet$ are known. We also call a state in $\mathcal{S}_g^\bullet$ a goal state. Since δ_g is total, we have $\Gamma_g(s) = \mathcal{A}_g$ for all $s \in \mathcal{S}_g$.

An episode starts at $t = 0$ with the agent at s_g° and terminates when the agent reaches a state in $\mathcal{S}_g^\bullet$. The interaction between the agent and its environment can be characterized by a feedback structure as illustrated in Figure 2. Let S_t and A_t denote the state where the agent is in, and the action taken by the agent, at time step t, respectively. We say that the state-action pair (S_t, A_t) is visited by the agent when at S_t the agent executes A_t according to its behavior policy π_b.

We do not need to be concerned with the exact form of π_b; we just require it to have the property (as discussed in Section 2) that for all $(s, a) \in \mathcal{S}_g \times \Gamma_g(s)$,

$$\pi_b(s, a) > 0 \tag{2}$$

if and only if $\delta_g(s, a)!$. Typical behavior policies, such as ϵ-greedy and softmax, have this property.

Upon executing A_t at time t the agent reaches the next state S_{t+1} as determined by δ_g at time $(t + 1)$ and receives a reward $R_{t+1} = \rho(S_t, A_t)$, where $\rho : \mathcal{S}_g \times \mathcal{A}_g \to \mathbb{R}$ is the reward function. The Q-value of a state-action pair $(s, a) \in \mathcal{S}_g \times \mathcal{A}_g$, defined in the standard way and denoted by $Q(s, a)$, is estimated iteratively in accordance with some pre-defined update rule based on the rewards received [51, 8]. We refer to (G, π_b) as the unconstrained learning process.

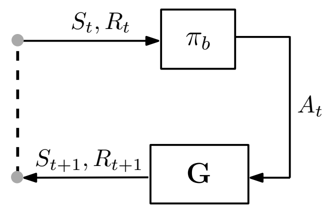

Figure 2: Unconstrained learning process. The dashed line indicates the advance of time; that is, $(\cdot)_{t+1}$ in the previous step becomes $(\cdot)_t$ in the current step.

3.2 Optimality in an unconstrained learning process

In practice, during learning the agent will carry out a finite number of episodes until the state-action values $Q(s, a)$ are considered to have converged to their optimum values, denoted by $Q^*(s, a)$, based on which an optimal policy can be obtained. Let $A_t^{(i)} \in \mathcal{A}_g$ denote the action taken by the agent at time step t during episode i, and $l_t^{(i)}$ denote the sequence of actions taken by the agent up to and including time step $(t-1)$, with $l_0^{(\cdot)} = \lambda$ (i.e., the empty string). Then we can write $l_t^{(i)} = A_0^{(i)} A_1^{(i)} \cdots A_{t-1}^{(i)} \in \mathcal{A}_g^\star$. By definition, $\left|l_t^{(i)}\right| = t$. Let $N \in \mathbb{N}_0$ be the total number of episodes in a learning process, and let n_i denote the total number of time steps in episode i. The language and the marked language generated by the agent over N episodes are $L = \cup_{i=1}^{N} \bar{l}_{n_i}^{(i)}$ and $L_m = \cup_{i=1}^{N} l_{n_i}^{(i)}$, respectively. For readability we sometimes omit the episode index (i), e.g., by writing l_t instead of $l_t^{(i)}$. With $\pi_b(s,a) > 0$ for all $(s,a) \in (\mathcal{S}_g - \mathcal{S}_g^\bullet) \times \mathcal{A}_g$, we have $L \rightarrow L(\mathrm{G})$ and $L_m \rightarrow L_m(\mathrm{G})$ as $N \rightarrow \infty$. We refer to $L(\mathrm{G})$ and $L_m(\mathrm{G})$ as the language and the marked language generated by (G, π_b), respectively, under the assumption that in practice a very large number of episodes are conducted.

With a properly designed update rule, the Q-values in an unconstrained learning system (G, π_b) will converge to their optimums under the Robbins-Monro conditions. To satisfy these conditions requires all state-action pairs be visited infinitely often by the agent during the learning process. In the context of this paper, which assumes pre-existence of a properly designed Q-value update rule, achieving optimality means meeting this requirement.

Proposition 3.1. Consider an unconstrained learning process (G, π_b). In any episode every state-action pairs $(s,a) \in (\mathcal{S}_g - \mathcal{S}_g^\bullet) \times \mathcal{A}_g$ has a non-zero probability of being visited by the agent.

Proof: For a state-action pair (s,a) to be visited by the agent in an unconstrained learning process (G, π_b), the following two conditions must be met. First, the agent must (starting from s_g°) reach the state s by executing some string $l \in \mathbb{C}_g(s)$. Second, the action a must be selected by the behavior policy π_b when the agent is at state s. Using the notation of probability languages presented in Section 2, the probability of a string $l \in \mathbb{C}_g(s)$ being generated can be expressed as $L_{\mathrm{G}}^p(l)$. Let $p_\pi(a|l)$ denote the probability that action a is taken by the agent at state s after the agent has executed the string l to reach s starting from s_g°, and let $p_u(s,a)$ denote the probability of a state-action pair (s,a) being visited by the agent. Then we can express this probability as

$$p_u(s,a) = p_\pi(a|l) \cdot L_{\mathrm{G}}^p(l) \tag{3}$$

We note that, by definition, $p_\pi(a|l) = \pi_b(s,a)$.

Since G is reachable and $\pi_b(s,a) > 0$ for all $(s,a) \in (\mathcal{S}_g - \mathcal{S}_g^\bullet) \times \Gamma_g(s)$, there exists a string l such that $l \in \mathbb{C}_g(s)$ for any $s \in \mathcal{S}_g$. In an episode i, the probability of a string $l_t^{(i)} \in \mathcal{C}_g(s)$ being generated is $L_{\mathrm{G}}^p(l_t^{(i)})$. With $A_t = a$,

$S_t = s$, and noting that $l_0^{(\cdot)}$ is the empty string (i.e., $l_0^{(\cdot)} = \lambda$), we have

$$\begin{aligned}
p_u(s,a) &\equiv p_u(S_t, A_t) \\
&= p_\pi(A_t | l_t^{(i)}) \cdot L_{\mathrm{G}}^p(l_t^{(i)}) \\
&= p_\pi(A_t | l_t^{(i)}) \cdot p_\pi(A_{t-1} | l_{t-1}^{(i)}) \cdot L_{\mathrm{G}}^p(l_{t-1}^{(i)}) \\
&\vdots \quad \text{(Continuing expansion till } L_{\mathrm{G}}^p(l_0^{(i)}) \text{ is reached)} \\
&= \underbrace{p_\pi(A_{t-1}|l_{t-1}) \cdots p_\pi(A_0|l_0^{(i)})}_{t \text{ terms}} \cdot \underbrace{L_{\mathrm{G}}^p(l_0)}_{L_{\mathrm{G}}^p(\lambda)=1} \\
&= \prod_{k=0}^{t-1} p_\pi(A_k|l_k) = \prod_{k=0}^{t-1} \pi_b(S_k, A_k) > 0 \qquad (4)
\end{aligned}$$

Hence, in any episode the probability of any steta-action pair $(s,a) \in (\mathcal{S}_g - \mathcal{S}_g^\bullet) \times \mathcal{A}_g$ being visited by the agent is non-zero. □

As the number of episodes goes to infinity, every state-action pairs in $(s,a) \in (\mathcal{S}_g - \mathcal{S}_g^\bullet) \times \mathcal{A}_g$ will being visited by the agent infinitely often.

4 Constrained Learning Behavior

To manipulate the exploration behavior of the agent is to impose on (G, π_b) an external mechanism that prevents (when necessary) the agent from taking certain action at a state, such that the language thus generated by the agent is restricted to a subset of $L(\mathrm{G})$. We call this mechanism a supervisor, denoted by Π, and refer to (G, π_b, Π) as a constrained learning process. Any mention of G hereafter is with reference to (G, π_b) or (G, π_b, Π) depending on the context, unless it is stated otherwise.

We refer to the set of actions available to the agent at a state as the admissible action set. In (G, π_b), the admissible action set at a state s is $\Gamma_g(s)$. In (G, π_b, Π), the supervisor may prevent the agent from taking a certain action at a state $s = \delta_g(s^\circ, l)$ by removing it from $\Gamma_g(s)$, thus reducing the admissible action set to a subset of $\Gamma_g(s)$. The supervisor decides which action (if any) to remove based on l. An action is said to be disabled if it is removed by the supervisor, and is said to be enabled otherwise. Only enabled actions belong to the admissible action set, which is denoted by $\Pi(l)$. Hence, $\Pi(l) \subseteq \Gamma_g(s) = \mathcal{A}_g$, and $\pi_b(s,a) = 0$ for all $a \notin \Pi(l)$.

Definition 4.1. A supervisor Π is a function from the language $L(\mathrm{G})$ to the power set of $\mathcal{A}_g$, i.e., $\Pi : L(\mathrm{G}) \to 2^{\mathcal{A}_g}$.

By definition a supervisor is any mechanism that can disable actions at a state. For example, a mechanism that randomly disables actions at a state is a supervisor. In Section 5, we will discuss the form of a supervisor (referred to as an "effective supervisor") that carries out such action-disablement selectively so that the agent under supervision will generate a specific language.

Remark 4.1. The concept of supervision by action-disablement and the realization of supervisor as automaton discussed in this section and in Section 5.1, respectviely, are adapted from the Supervisory Control Theory that focuses on the control of discrete-event systems [46]. Central to this adaptation is the establishment of an extended feedback-control structure (involving G, π_b and Π) in the context of the class of reinforcement learning problems under investigation.

4.1 Structure and characterization of supervision

The interaction between Π and (G, π_b) in a constrained learning process is illustrated in Figure 3. Suppose that the agent, starting from the initial state $S_0 = s_g^\circ$ at time step $t = 0$ and following some action sequence $l_t \in L(\mathrm{G})$, reaches the state S_t at time step $t > 0$, i.e., $S_t = \delta_g(S_0, l_t)$. At time step t the supervisor Π generates (based on l_t) the admissible action set $\Pi(l_t)$, from which the behavior policy π_b selects one action (if $\Pi(l_t)$ is non-empty) that the agent is to execute. Upon taking an action A_t at time step t, the agent reaches the next state S_{t+1} and receives a reward R_{t+1}. The supervisor then generates, with $l_{t+1} = l_t\, A_t$, the next admissible action set $\Pi(l_{t+1})$ from which π_b selects the next action. This cycle repeats until the episode terminates.

Under the structure as illustrated in Figure 3, the supervisor Π exerts a dynamic feedback control on (G, π_b). It is dynamic because the decision on whether to disable an action at a state $s = \delta_g(s_g^\circ, l)$ depends not on s itself but on how s is reached; that is, for two different action sequences $l_1, l_2 \in \mathbb{C}_g(s)$, Π may disable an action a when s is reached via l_1 but may not do so when s is reached via $l_2 \neq l_1$.

Definition 4.2. The language generated by the agent under the supervision of Π, denoted by $L(\Pi/\mathrm{G})$, is defined recursively as follows:

(1) $\lambda \in L(\Pi/\mathrm{G})$

(2) $\Big[l \in L(\Pi/\mathrm{G}) \text{ and } la \in L(\mathrm{G}) \text{ and } a \in \Pi(l) \Big] \Leftrightarrow la \in L(\Pi/\mathrm{G})$

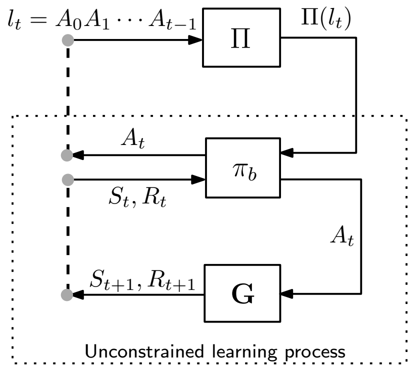

Figure 3: Feedback structure of a constrained learning process. A dashed line indicates the advance of time; that is, $(\cdot)_{t+1}$ in the previous step becomes $(\cdot)_t$ in the current step.

By definition, $L(\Pi/\text{G})$ is prefix-closed, i.e., $L(\Pi/\text{G}) = \overline{L(\Pi/\text{G})}$. If $\Pi(l) \subset \Gamma_g(s)$ for some $s \in \left(\mathcal{S}_g - \mathcal{S}_g^{\bullet}\right)$, then the effect of Π is to restrict the behavior of the agent to some sublanguage of $L(\text{G})$, i.e., $L(\Pi/\text{G}) \subset L(\text{G})$.

4.2 Specification language

We consider only desired behavior that can be expressed as a regular and prefix-closed sublanguage of $L(\text{G})$. We call such a language a specification language and denote it by $\mathcal{L}$, i.e., $\mathcal{L} \subset L(\text{G})$. Since $\mathcal{L}$ is regular, we can construct a finite state automaton, denoted by H, that recognizes $\overline{\mathcal{L}}$ (i.e., the prefix-closure of $\mathcal{L}$); that is $L(\text{H}) = \overline{\mathcal{L}}$. (In fact, H also recognizes $\mathcal{L}$ since $\mathcal{L}$ is assumed to be prefixed-closed, i.e., $\mathcal{L} = \overline{\mathcal{L}}$.) We will show in Section 5 that under the supervision of Π the behavior of the agent, i.e., the language $L(\Pi/\text{G})$, is the same as the desired behavior, i.e., $L(\Pi/\text{G}) = L(\text{H}) = \mathcal{L}$.

In practice, we can first express the desired agent behavior in natural language, then encode the logical relationships implied in that behavior in the transition structure of H. Methods of constructing H for some typical logical relationships are discussed in [11]. Such an automaton plays a key role in the realization of an effective supervisor, as discussed in Section 5.1.

Example 4.1. Consider an unconstrained learning process (G_1, π_{b1}) in which the agent is to traverse from an initial state to a goal state in a two-dimensional grid-world. At a state the agent can take one of the four actions to move up (denoted by a_1), right (a_2), down (a_3), and left (a_4), resulting in the agent reaching deterministically the adjacent state in the direction of the intended action, or remaining in the same state if it collides with a border of the grid. Suppose that it is desirable to prevent the agent from oscillating between two adjacent (non-bordered) states by taking the action sequence a_1a_3 or a_2a_4. The specification language for this desired behavior consists of all strings in $L(\mathrm{G}_1)$ except those that contain a_1a_3 or a_2a_4 as a substring, i.e., $\mathcal{L}_1 = L(\mathrm{G}_1) - \mathcal{L}'_1$, where $\mathcal{L}'_1 := \{l \in L(\mathrm{G}_1) \,|\, a_1a_3 \prec l \text{ or } a_2a_4 \prec l\}$. An automaton that generates $\mathcal{L}_1$ is shown in Figure 4. By construction, $L(\mathrm{H}_1) = \mathcal{L}_1$ is regular and prefix-closed.

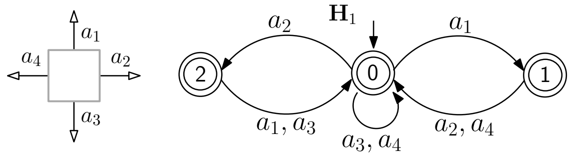

Figure 4: Automaton H_1 that generates the language $\mathcal{L}_1$. The entering arrow indicates the initial state; a double circle indicates a marked state.

5 Effectiveness of Supervisor and Optimality Preservation

Definition 5.1. Consider an unconstrained learning process (G, π_b) and a prefix-closed language $\mathcal{L} \subset L(\mathrm{G})$. A supervisor Π is effective with respect to $\mathcal{L}$ if $L(\Pi/\mathrm{G}) = \overline{\mathcal{L}}$.

In this section, we prove that, given a specification language $\mathcal{L} \subset L(\mathrm{G})$, an effective supervisor exists. We then establish a necessary and sufficient condition under which an effective supervisor is optimality-preserving.

5.1 Existence and realization of supervisor

Lemma 5.1. For a given unconstrained learning process (G, π_b) and a prefix-closed language $\mathcal{L} \subset L(\mathrm{G})$, an effective supervisor exists.

Proof: Since $\mathcal{L}$ is prefix-closed, we have $\mathcal{L} = \overline{\mathcal{L}}$. Define a supervisor

$$\Pi(l) = \left\{a \in \mathcal{A}_g \,|\, l \in L(\mathrm{G}) \text{ and } la \in \overline{\mathcal{L}}\right\} \tag{5}$$

We will carry out the proof by induction on the length of strings of $L(\Pi/\mathrm{G})$ and $\overline{\mathcal{L}}$ to show that, with Π as defined above, $l \in L(\Pi/\mathrm{G})$ if and only if $l \in \overline{\mathcal{L}}$. The base case of $l = \lambda$ is trivial since λ belongs to both languages by definition. The hypothesis is that $l \in L(\Pi/\mathrm{G})$ if and only if $l \in \overline{\mathcal{L}}$. The inductive step involves strings of the form la, with $a \in \mathcal{A}_g$. For the IF case, we have $la \in \overline{\mathcal{L}}$. Since $\overline{\mathcal{L}}$ is prefix-closed, we have $l \in \overline{\mathcal{L}}$; consequently, $l \in L(\Pi/\mathrm{G})$ (from the hypothesis). Since $l \in L(\Pi/\mathrm{G}) \subset L(\mathrm{G})$ and $la \in \overline{\mathcal{L}}$, from Equation (5) we have $a \in \Pi(l)$. Hence, $la \in L(\Pi/\mathrm{G})$. For the ONLY-IF case, $la \in L(\Pi/\mathrm{G})$ implies that $l \in L(\Pi/\mathrm{G})$ (since $L(\Pi/\mathrm{G})$ is prefix-closed), which by the hypothesis implies that $l \in \overline{\mathcal{L}}$. Now $l \in L(\Pi/\mathrm{G})$ and $la \in L(\Pi/\mathrm{G})$ implies that $a \in \Pi(l)$. Therefore, $la \in \overline{\mathcal{L}}$. $\square$

The function Π can be represented as an automaton that 'reads' a string l and generates the admissible action set $\Pi(l)$. When operating in the structure illustrated in Figure 3, the interaction between H and (G, π_b) can be implemented via the product H$\|$G; that is, the language generated by the agent under the supervision of Π, i.e., $L(\Pi/\mathrm{G})$, is the same as the language generated by H$\|$G [54, 11]. Specifically, the automaton H is said to be a realization of Π if $L(\mathrm{H}\|\mathrm{G}) = L(\Pi/\mathrm{G})$ and $L_m(\mathrm{H}\|\mathrm{G}) = L_m(\Pi/\mathrm{G})$.

Remark 5.1. If during an episode $\Pi(l_t)$ is empty at time step t for a string $l_t \in C_g\,(s) \cap L\,(\Pi/\mathrm{G})$, then no action at the state s is enabled by the supervisor. Consequently, no action is available for selection by the behavior policy π_b at s at time step t. If s is not a marked (i.e., goal) state, then the agent will deadlock at s, and the episode ends prematurely at time step t. The behavior of the agent up to time step t still meets the specification $\mathcal{L}$.

Proposition 5.1. A trim automaton H (with $\mathcal{A}_h = \mathcal{A}_g$ and all states marked) is a realization of the supervisor Π as defined in Equation (5) if $L(\mathrm{H}) = \mathcal{L}$, where $\mathcal{L} = \overline{\mathcal{L}} \subset L(\mathrm{G})$.

Proof: Since H is trim and $L(\mathrm{H}) = \mathcal{L}$ is prefix-closed, we have $L_m(\mathrm{H}) = L(\mathrm{H}) = \overline{\mathcal{L}}$. Now $L(\mathrm{H}\|\mathrm{G}) = L(\mathrm{H}) \cap L(\mathrm{G}) = \overline{\mathcal{L}} \cap L(\mathrm{G})$. Since Π is effective, we have $L(\mathrm{H}\|\mathrm{G}) = \overline{\mathcal{L}} = L(\Pi/\mathrm{G})$. For the marked language $L_m(\mathrm{H}\|\mathrm{G})$, we have $L_m(\mathrm{H}\|\mathrm{G}) = L_m(\mathrm{H}) \cap L_m(\mathrm{G}) = \overline{\mathcal{L}} \cap L_m(\mathrm{G}) = L(\Pi/\mathrm{G}) \cap L_m(\mathrm{G}) = L_m(\Pi/\mathrm{G})$. The last step reflects the fact that, because all states of H are marked, marking in (G, π_b, Π) is done based on $\mathcal{S}_g^\bullet$ and does not involve Π. $\square$

Remark 5.2. The proofs of Lemma 5.1 and Proposition 5.1 imply that, given a prefix-closed language $\mathcal{L} = L(\mathrm{H}) \subset L(\mathrm{G})$, an effective supervisor can be realized using the automaton H without any explicit knowledge of the transition structure of G.

5.2 Optimality preservation

The existence of an effective supervisor (established in Lemma 5.1) naturally gives rise to the question of whether, with its behavior restricted by Π to $\overline{\mathcal{L}} \subset$

$L(\mathrm{G})$, the agent is still able to find the intrinsic optimums of the Q-values; that is, whether optimality in (G, π_b) is preserved under the supervision of Π. Recall that a requirement for satisfying the Robbins-Monro conditions in (G, π_b), as discussed in Section 3.2, is that all state-action pairs be visited infinitely often; that is, the probability of any action in $\mathcal{A}_g$ being taken by the agent at any state in $(\mathcal{S}_g - \mathcal{S}_g^\bullet)$ must be non-zero. Optimality preservation means that this requirement is satisfied in (G, π_b, Π).

For a state-action pair (s, a) to be visited by the agent in (G, π_b, Π), the following three conditions must be met. First, the agent must (starting from s_g°) reach the state s by executing some string $l \in \mathbb{C}_g(s) \cap L(\Pi/\mathrm{G})$. Second, the action a must be enabled by the supervisor Π at s. Third, the action a must be selected by the behavior policy π_b when the agent is at state s. Suppose that a supervisor Π is realized by an automaton H. Then we have $L(\Pi/\mathrm{G}) = L(\mathrm{H}\|\mathrm{G})$. Recall from the proof of Proposition 3.1 that $p_\pi(a|l)$ denotes the probability that the action a is taken by the agent after the agent has executed the action sequence l to reach s starting from s_g°. Now let $p_\Pi(a|l)$ denote the probability that the action a is enabled by Π after the agent has executed l to reach s starting from s_g°. Using the notation of probability languages presented in Section 2, the probability of a string $l \in \mathbb{C}_g(s) \cap L(\Pi/\mathrm{G})$ being generated by the agent under the supervision of Π can be expressed as $L^p_{\mathrm{H}\|\mathrm{G}}(l)$. Then the probability of a state-action pair (s, a) being visited by the agent under the supervision of Π, denoted by $p(a|s)$, can be expressed as

$$p(a|s) = p_\pi(a|l) \cdot p_\Pi(a|l) \cdot L^p_{\mathrm{H}\|\mathrm{G}}(l) \tag{6}$$

Lemma 5.2. $L^p_{\mathrm{H}\|\mathrm{G}}(l) > 0$.

This lemma is needed in the proof of the main result stated in Theorem 5.1. The poof for Lemma 5.2 is presented in Appendix A; it follows the same line of argument based on the expansion of l (in terms of its individual constituent actions taken by the agent at each time step) as shown in the proof of Proposition 3.1.

Definition 5.2. A supervisor Π in a constrained learning process (G, π_b, Π) is said to be optimality-preserving if $p(a|s) > 0$ for all $(s, a) \in (\mathcal{S}_g - \mathcal{S}_g^\bullet) \times \mathcal{A}_g$.

By definition a supervisor can be optimality-preserving without being effective. A trivial example is a supervisor that does not disable any action at any state. In the context of this paper, whether an effective supervisor is optimality-preserving depends on the desired behavior, as is illustrated in Example 1.1 in Section 1.1.

We next establish a condition for a given effective supervisor to be optimality-preserving in terms of a property of the specification language $\mathcal{L}$. For this purpose, we introduce the notion of an automaton being 'covered' by some sublanguage of the language generated by that automaton.

Definition 5.3. Let $\mathcal{L} \subset L(\mathrm{G})$. We say that $\mathcal{L}$ covers G if, for any $(s, a) \in (\mathcal{S}_g - \mathcal{S}_g^{\bullet}) \times \mathcal{A}_g$, there exists a string $l \in \mathbb{C}_g(s) \cap \mathcal{L}$ such that $la \in \mathcal{L}$.

Informally, a language $\mathcal{L}$ covering an automaton G means that for any non-marked state s in G and any action $a \in \mathcal{A}_g$, we can always find a string l in $\mathcal{L}$ satisfying the following two properties: (1) starting from the initial state s_g° the state s can be reached by following l, and (2) the string la also belongs to $\mathcal{L}$. It is possible that, of all the strings that have property 1, some do not have property 2. As long as for each state-action pair there is at least one string in the specification language that satisfies both properties, then the coverage holds. In the context of a constrained learning process, this notion of coverage means that, in order to satisfy the specification, the supervisor may block an action at a state if that state is reached via some action sequence but will not do so if the same state is reached via a different action sequence. This notion of coverage, in conjunction with the main result presented below, will be illustrated in two scenarios in Section 5.4.

We now present the main result.

Theorem 5.1. An effective supervisor Π is optimality-preserving if and only if $\mathcal{L}$ covers G.

Proof: Since Π is effective, we have $L(\Pi/\mathrm{G}) = \mathcal{L}$. We first prove the IF case. Since $\mathcal{L}$ covers G, there exists a string $l \in \mathbb{C}_g(s) \cap \mathcal{L} = \mathbb{C}_g(s) \cap L(\Pi/\mathrm{G})$ such that $la \in L(\Pi/\mathrm{G})$ for any $(s, a) \in (\mathcal{S}_g - \mathcal{S}_g^{\bullet}) \times \mathcal{A}_g$. Now from Equation (6), we have $p(a|s) = p_{\pi}(a|l) \cdot p_{\Pi}(a|\,l) \cdot L^{p}_{\mathrm{H}\|\mathrm{G}}(l)$. Since $l \in L(\Pi/\mathrm{G})$ and $la \in L(\Pi/\mathrm{G})$, we have $a \in \Pi(l)$, i.e., a is enabled by Π at s. So $p_{\Pi}(a|\,l) = 1$ and $\pi_b(s, a) > 0$. By definition $p_{\pi}(a|l) = \pi_b(s, a)$; so $p_{\pi}(a|l) > 0$. From Lemma 5.2, we have $L^{p}_{\mathrm{H}\|\mathrm{G}}(l) > 0$. Hence, $p(a|s) > 0$.

We next prove the ONY-IF case. Specifically, we will show that, under an effective Π if $p(a|s) > 0$ for any $(s, a) \in (\mathcal{S}_g - \mathcal{S}_g^{\bullet}) \times \mathcal{A}_g$, then there exists a string $l \in \mathbb{C}_g(s) \cap L(\Pi/\mathrm{G})$ such that $la \in L(\Pi/\mathrm{G})$. Consider any $(s, a) \in (\mathcal{S}_g - \mathcal{S}_g^{\bullet}) \times \mathcal{A}_g$ with some $l \in \mathbb{C}_g(s)$. From Equation (6), $p(a|s) > 0$ implies that there exists a string $l \in \mathbb{C}_g(s)$ such that $p_{\pi}(a|l) > 0$, $p_{\Pi}(a|\,l) > 0$, and $L^{p}_{\mathrm{H}\|\mathrm{G}}(l) > 0$. From Equation (9) in the appendix, we have $L^{p}_{\mathrm{H}\|\mathrm{G}}(l) \equiv L^{p}_{\mathrm{H}\|\mathrm{G}}(l_t) = \prod_{k=0}^{t-1} p_{\pi}(A_k|l_k) \cdot p_{\Pi}(A_k|l_k)$. So $L^{p}_{\mathrm{H}\|\mathrm{G}}(l) > 0$ implies that $p_{\Pi}(A_k|\,l_k) > 0$ (which in turn implies that $p_{\Pi}(A_k|\,l_k) = 1$) for all

$k \in [\![0, t-1]\!]$. Hence, $A_k \in \Pi(l_k)$ for all $k \in [\![0, t-1]\!]$. Consequently, we have $l_t \equiv l \in L(\Pi/\mathrm{G})$. We next show that $la \in L(\Pi/\mathrm{G})$. Since $p_\Pi(a|l) > 0$ implies that $p_\Pi(a|l) = 1$, we have $a \in \Pi(l)$. Therefore, $la \in L(\Pi/\mathrm{G}) = \mathcal{L}$. □

Intuitively, if we think of the agent's movement through the state space as being described by a sequence of state-action pairs, e.g., $S_0A_0S_1A_1 \cdots S_tA_t$, then an optimality-preserving supervisor will permit at least one trajectory to pass through any state-action pair (where the state is not a marked state). The existence of such permitted trajectories depends on whether the specification covers G.

5.3 Deciding whether $\mathcal{L} \subset \mathrm{G}$ covers G

We next present a sufficient condition for deciding whether a given $\mathcal{L} \subset L(\mathrm{G})$ covers G. This condition is based on the product of G and an automaton H that generates $\mathcal{L}$, i.e., $L(\mathrm{H}) = \mathcal{L}$, and involves the notion of the visitablity of a state in G. Let $\mathrm{M} = \mathrm{H} \| \mathrm{G}$. Define a function $\Omega_m : \mathcal{S}_g \to 2^{\mathcal{S}_m}$ (with $\mathcal{S}_m = \mathcal{S}_h \times \mathcal{S}_g$) that maps a state $s_g \in \mathcal{S}_g$ to a set that contains just the states in M whose second component is s_g, i.e., $\Omega_m(s_g) := \{(a, b) \mid a \in \mathcal{S}_h, b \in \mathcal{S}_g, \text{ and } b = s_g\}$. Consider in M two states (s_h, s_g) and (s'_h, s_g) reached by two different strings l and l' that both belong to $\mathbb{C}_g(s_g)$. By definition, (s_h, s_g) and (s'_h, s_g) belong to $\Omega_m(s_g)$. It is possible that an action $a \in \mathcal{A}_g$ is disabled by Π at (s_h, s_g) but enabled at (s'_h, s_g). If after a number of visits by the agent to the states in $\Omega(s_g)$ all actions in $\mathcal{A}_g$ have been enabled (at least once) by Π, then all state-action pairs $(s_g, a)|_{a \in \mathcal{A}_g}$ have a non-zero probability of being visited by the agent. We characterize this property as the visitability of a state.

Definition 5.4. A state $s_g \in (\mathcal{S}_g - \mathcal{S}_g^\bullet)$ in G is visitable with respect to M if

$$\bigcup_{(s_h, s_g) \in \Omega_m(s_g)} \Gamma_m((s_h, s_g)) = \mathcal{A}_g \tag{7}$$

Lemma 5.3. If a state $s_g \in (\mathcal{S}_g - \mathcal{S}_g^\bullet)$ is visitable with respect to M, then for any $a \in \mathcal{A}_g$ there exits a state $(s_h, s_g) \in \Omega_m(s_g)$ such that $a \in \Gamma_m((s_h, s_g))$.

Proof: This lemma follows directly from Definition 5.4. Suppose that there are n states in M corresponding to s_g, i.e.,

$$\Omega_m(s_g) = \{(s_{h_1}, s_g), (s_{h_2}, s_g), \ldots, (s_{h_n}, s_g)\}$$

Equation (7) can then be expressed as

$$\bigcup_{(s_h,s_g)\in\Omega_m(s_g)} \Gamma_m((s_h,s_g)) = \Gamma_m((s_{h_1},s_g)) \cup \cdots \cup \Gamma_m((s_{h_n},s_g)) = \mathcal{A}_g \quad (8)$$

Any $a \in \mathcal{A}_g$ must belong to at least one of the active action sets $\Gamma_m(\cdot)$ in Equation (8). $\square$

Definition 5.5. An automaton G is visitable with respect to M if every state in $(\mathcal{S}_g - \mathcal{S}_g^\bullet)$ is visitable with respect to M.

We now state a condition for deciding whether a given $\mathcal{L} \subset L(\mathrm{G})$ covers G.

Theorem 5.2. If G is visitable with respect to M, then $\mathcal{L}$ covers G.

Proof: Consider any state-action pair $(s_g, a) \in (\mathcal{S}_g - \mathcal{S}_g^\bullet) \times \mathcal{A}_g$. From Lemma 5.3, for any $a \in \mathcal{A}_g$ we have $a \in \Gamma_m((s_h, s_g))$ for some $(s_h, s_g) \in \Omega_m(s_g)$. Since M is reachable by construction, there exists a string $l \in \mathbb{C}_m((s_h, s_g))$. Consequently, $la \in L(\mathrm{M}) = L(\mathrm{H}) = \mathcal{L}$. Since $(s_h, s_g) \in \Omega_m(s_g)$, so $l \in \mathbb{C}_m((s_h, s_g))$ implies that $l \in \mathbb{C}_g(s_g)$. Therefore, we have $l \in \mathbb{C}_g(s_g) \cap \mathcal{L}$. Since $l \in \mathbb{C}_g(s_g) \cap \mathcal{L}$ and $la \in \mathcal{L}$ for any $(s_g, a) \in (\mathcal{S}_g - \mathcal{S}_g^\bullet) \times \mathcal{A}_g$, we conclude that $\mathcal{L}$ covers G. $\square$

Theorem 5.2 and Lemma 5.3 provide a method for checking whether a given $\mathcal{L}$ covers G: For each $s_g \in (\mathcal{S}_g - \mathcal{S}_g^\bullet)$, determine $\Omega_m(s_g)$. If Equation (7) is satisfied for all $s_g \in (\mathcal{S}_g - \mathcal{S}_g^\bullet)$, then $\mathcal{L}$ covers G.

5.4 Illustrative Scenarios

In this section we illustrate the concept of a sublanguage of an automaton covering that automaton. We use two scenarios involving a 4×4 grid environment G with the transition characteristics as described in Example 4.1. The agent is to find an optimal policy for it to reach the marked state at the lower right corner of the grid, starting from the initial state at the top left corner. The first scenario concerns a specification language that covers G; the second concerns one that does not.

In the first scenario, the agent is prohibited from taking two right (i.e., a_2) actions consecutively. An automaton H_1 that generates a prefix-closed specification language $\mathcal{L}_1 = L(\mathrm{H}_1) \subset L(\mathrm{G})$ satisfying this requirement is shown in Figure 5.

It can be checked that G is visitable with respect to $\mathrm{M}_1 = \mathrm{H}_1 \| \mathrm{G}$. Due to the nature of the specification language $\mathcal{L}_1$, only the state-action pairs

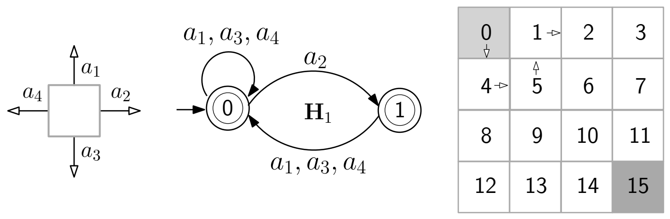

Figure 5: Left: Actions at a state. Middle: Automaton H_1 representing the requirement that the agent is prohibited from taking two consecutive a_2 actions. Right: A 4×4 grid environment, with the initial state at 0 and the marked state at 15. The four small arrows in the grid represent the action sequence $a_3\, a_2\, a_1\, a_2$.

involving states that are not next to the left border and involving action a_2 need to be checked — since the transition structure in G does not allow a state next to the left border to be reached by an a_2 action. Take the state-action pair $(1, a_2)$ for instance. Although the agent is not allowed to take a_2 at state 1 after taking a_2 at state 0 (i.e., a_2a_2 cannot be a substring of any string in $\mathcal{L}_1$), it can still take action a_2 at state 1 via the sequence $a_3\, a_2\, a_1\, a_2$ (which is a substring of a string in $\mathcal{L}_1$), as is illustrated by the small arrows in the grid in Figure 5. So the state-action pair $(1, a_2)$ in G is visitable with respect to M_1. This can be verified by examining the transition structure of M_1. Table 1 shows the actions that are active at the two composite states involving state 1 of G, namely, $(0, 1)$ and $(1, 1)$. Although a_2 is not active at $(1, 1)$, it is active at $(0, 1)$. Therefore, state 1 of G is visitable with respect to M_1. The same analysis can be done for all the states in G that are not next to the left border to ascertain that G is visitable with respect to M_1. So $\mathcal{L}_1$ covers G. Consequently, an effective supervisor as defined in Equation (5) with respect to $\mathcal{L}_1$ is optimality-preserving.

In the second scenario, the requirement is that the agent can take an a_2 action only immediately after having taken an a_3 action. An automaton H_2 that generates a prefix-closed specification language $\mathcal{L}_2 = L(\mathrm{H}_2) \subset L(\mathrm{G})$ satisfying this requirement is shown in Figure 6.

Since there is no downward action (i.e., a_3) that brings the agent to states 0, 1, 2, and 3 in G, we can conclude by inspection that to comply with the requirement the agent must not move to the right at those states, as is indicated by the missing right arrow at those states in Figure 6. This conclusion can be verified by examining the transition structure of $\mathrm{M}_2 = \mathrm{H}_2 \| \mathrm{G}$. We consider the state-action pair $(1, a_2)$ in G as an illustration. In

Table 1: Active actions in M_1 involving state 1 of G.

H_1		G		$M_1 = H_1 \| G$	
s_{h_1}	a_{h_1}	s_g	a_g	s_{m_1}	a_{m_1}
0	a_1	1	a_1	$(0,1)$	a_1
0	a_2	1	a_2	$(0,1)$	a_2
0	a_3	1	a_3	$(0,1)$	a_3
0	a_4	1	a_4	$(0,1)$	a_4
1	a_1	1	a_1	$(1,1)$	a_1
1	$-$	1	a_2	$(1,1)$	$-$
1	a_3	1	a_3	$(1,1)$	a_3
1	a_4	1	a_4	$(1,1)$	a_4

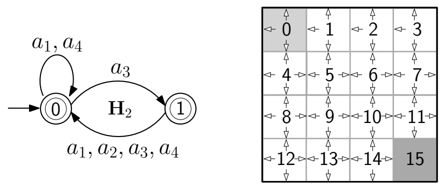

Figure 6: Left: Automaton H_2 representing the requirement that the agent can take action a_2 only immediately after having taken a_3. Right: Behavior of the agent that satisfies $\mathcal{L}_2$.

$\mathcal{S}_{h_2} \times \mathcal{S}_g$ the only two $(s_{h_2},\, s_g)$ pairs that involve state 1 of G are $(0,1)$ and $(1,1)$. However, the pair $(1,1)$ is not a state in M_2 because the only way for H_2 to reach its own state 1 is via an a_3 action but, due to the structure of G, state 1 in G cannot be reached by an a_3 action. Consequently, of these two pairs only $(0,1)$ is a state in M_2. Table 2 shows the actions that are active at $(0,1)$. Since a_2 is not active at state 0 of H_2, it cannot be active at $(0,1)$. Therefore, state 1 of G is not visitable with respect to M_2. Similar arguments can be made concerning the state-action pairs $(0,a_2)$, $(2,a_2)$, and $(3,a_2)$ of G. We thus conclude that $\mathcal{L}_2$ does not cover G.

6 Discussion

In this section we first discuss the implications and limitations of our proposed approach. We then elaborate on a possible direction to extend the current work in the context of the theory of supervisory control of discrete-event systems.

Table 2: Active actions in M_2 involving state 1 of G.

H_2		G		$M_2 = H_2 \| G$	
s_{h_2}	a_{h_2}	s_g	a_g	s_{m_2}	a_{m_2}
0	a_1	1	a_1	$(0,1)$	a_1
0	$-$	1	a_2	$(0,1)$	$-$
0	a_3	1	a_3	$(0,1)$	a_3
0	a_4	1	a_4	$(0,1)$	a_4

The existence and realization of an effective and optimality-preserving supervisor, established in Lemma 5.1 and Theorem 5.1, suggest a rigorous approach for manipulating the behavior of the agent during a reinforcement-learning process. Under this approach, achieving desired agent behavior is decoupled from achieving convergence of the learning process. Consequently, the choice of an update rule for the Q-values is independent of the design of a supervisor that restricts the agent to certain prescribed behavior. This decoupling makes possible the application of this approach in various scenarios that share an underlying theme of manipulating the behavior of the agent.

The applicability of our proposed approach is limited by two main factors. The first concerns the assumption that a model of the environment G is known. Ideally we seek a supervisor that is both effective and optimality-preserving, so that we can manipulate the behavior of the agent without interfering with the determination of the intrinsic optimums; moreover, we would like to be able to ascertain the existence of such a supervisor a priori. This is possible when a deterministic environment model G is known, as shown in the proofs of Lemma 5.1 and Theorem 5.1. In the model-free setting where G is unknown, even though we are still able to construct an effective supervisor in the form of an automaton H that generates $\mathcal{L}$ (see Remark 5.2), the method presented in Section 5.2 is not applicable in checking a priori whether this effective supervisor is optimality-preserving, because doing so involves the computation of H$\|$G. A possible way of relaxing this assumption is to apply the framework of stochastic automata and probabilistic languages (e.g., [18, 32], so as to obtain some result concerning optimality preservation in the probabilistic sense.

The second limiting factor concerns the nature of supervision. In this paper the supervisor is assumed to be omnipotent in the sense that whenever it decides to disable an action $a \in \mathcal{A}_g$ at a state $s \in \mathcal{S}_g$, the behavior policy π_b always complies with that decision, i.e., $\pi_b(s, a) = 0$. A more interesting control-theoretic issue arises if such compliance is not total with respect to

$\mathcal{A}_g$; that is, the supervisor is able to disable some but not all actions in $\mathcal{A}_g$. The question of how to deal with this issue is addressed in the Supervisory Control Theory [46, 54, 11], and thus serves as a motivation to extend the current work in the context of that theory.

In the Theory of Supervisory Control, an action is considered controllable if it can be disabled by the supervisor, and uncontrollable otherwise. The set of events (i.e., actions in this paper), denoted by Σ, is partitioned into two disjoint subsets Σ_c and Σ_u, with Σ_c containing events that are controllable and Σ_u uncontrollable. Under this arrangement, the aim of supervisory control is to ensure, without disabling any uncontrollable events, that the behavior of the system under supervisory control stays within a specification language. The scope of supervision as proposed in this paper in the context of reinforcement learning (in particular Lemma 5.1) covers a special case of that arrangement, where all actions in $\mathcal{A}_g$ are assumed to be controllable, i.e., $\Sigma = \Sigma_c = \mathcal{A}_g$. A particularly meaningful way to extend the scope of supervision is to remove this assumption, so that pertinent control-theoretic properties (such as the controllability of the behavior of the agent) can be investigated. This extension would represent an important step in establishing a formal connection between the Supervisory Control Theory and a class of reinforcement-learning systems.

7 Conclusion

We have introduced the concept of constrained exploration in reinforcement learning with optimality preservation in terms of manipulating the behavior of the agent to meet some specification during the learning process. We have investigated this concept in the context of the theory of automata and formal languages by (i) establishing a feedback-control structure that models the dynamics of the unconstrained learning process, (ii) extending this structure by adding a supervisor to ensure that the behavior of the agent meets if possible a given specification, and (iii) establishing a necessary and sufficient condition (on the specification language) for optimality to be preserved under supervision.

The work reported in this paper has resulted in a formal method for manipulating the behavior of the agent in a class of reinforcement-learning systems. It demonstrates the utility and the prospect of studying reinforcement learning in the context of the theories of discrete-event systems, automata and formal languages.

Appedix A: Proof of Lemma 5.2

In Lemma 5.2, the claim is that, for a state s in a constrained learning system (G, π_b, Π), we have $L^p_{\mathrm{H}\|\mathrm{G}}(l) > 0$ with $l \in \mathbb{C}_g(s) \cap L(\Pi/\mathrm{G})$. The proof below follows the same line of argument based on the expansion of l as shown in the proof of Proposition 3.1. Recall that $|l| = t$. We first express $L^p_{\mathrm{H}\|\mathrm{G}}(l)$ in terms of the constituent actions of l up to and including time step $(t-1)$, i.e., $l \equiv l_t = A_0 A_1 \cdots A_{t-1}$. Now,

$$\begin{aligned}
L^p_{\mathrm{H}\|\mathrm{G}}(l) &\equiv L^p_{\mathrm{H}\|\mathrm{G}}(l_t) \\
&= p_\pi(A_{t-1}|\, l_{t-1}) \cdot p_\Pi(A_{t-1}|\, l_{t-1}) \cdot L^p_{\mathrm{H}\|\mathrm{G}}(l_{t-1}) \\
&= p_\pi(A_{t-1}|l_{t-1}) \cdot p_\Pi(A_{t-1}|\, l_{t-1}) \cdot p_\pi(A_{t-1}|l_{t-2}) \cdot p_\Pi(A_{t-1}|\, l_{t-2}) \cdot L^p_{\mathrm{H}\|\mathrm{G}}(l_{t-2}) \\
&\vdots \quad \text{(Continuing expansion till } L^p_{\mathrm{H}\|\mathrm{G}}(l_0) \text{ is reached)} \\
&= \underbrace{p_\pi(A_{t-1}|l_{t-1}) \cdots p_\pi(A_0|l_0)}_{t \text{ terms}} \cdot \underbrace{p_\Pi(A_{t-1}|l_{t-1}) \cdots p_\Pi(A_0|l_0)}_{t \text{ terms}} \cdot \underbrace{L^p_{\mathrm{H}\|\mathrm{G}}(l_0)}_{L^p_{\mathrm{H}\|\mathrm{G}}(\lambda)=1} \\
&= \prod_{k=0}^{t-1} p_\pi(A_k|l_k) \cdot p_\Pi(A_k|l_k) \qquad (9)
\end{aligned}$$

Since $l \in L(\Pi/\mathrm{G})$, we have $p_\Pi(A_k|l_k) = 1$; that is, $A_k \in \Pi(l_k)$ for all $k \in [\![0, t-1]\!]$. By definition $p_\pi(A_k|l_k) = \pi_b(S_k, A_k)$, where $S_k = \delta_g(s_g^\circ, l_k)$. Since $A_k \in \Pi(l_k)$ and $\pi_b(S_k, A_k) > 0$ if $A_k \in \Pi(l_k)$, we have $p_\pi(A_k|l_k) > 0$ for all $k \in [\![0, t-1]\!]$. Therefore, $L^p_{\mathrm{H}\|\mathrm{G}}(l_t) > 0$. □

References

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In Proceedings of the Twenty-First International Conference on Machine Learning, Banff Alberta Canada, 2004.

[2] David Abel, John Salvatier, Andreas Stuhlmller, and Owain Evans. Agent-agnostic human-in-the-loop reinforcement learning. arXiv preprint arXiv:1701.04079, 2017.

[3] Mohammed Alshiekh, Roderick Bloem, Ruediger Ehlers, Bettina Knighofer, Scott Niekum, and Ufuk Topcu. Safe Reinforcement Learning via Shielding, September 2017. arXiv:1708.08611.

[4] Ofra Amir, Ece Kamar, Andrey Kolobov, and Barbara J Grosz. Interactive teaching strategies for agent training. In Proceedings of the

Twenty-Fifth international joint conference on Artificial Intelligence (IJ-CAI'16), 2016.

[5] Fahiem Bacchus, Craig Boutilier, and Adam Grove. Rewarding behaviors. In Proceedings of the National Conference on Artificial Intelligence, pages 1160–1167, 1996.

[6] Osbert Bastani. Safe Reinforcement Learning with Nonlinear Dynamics via Model Predictive Shielding. In 2021 American Control Conference (ACC), pages 3488–3494, May 2021.

[7] Felix Berkenkamp, Matteo Turchetta, Angela P Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. Advances in Neural Information Processing Systems, 2:909–919, 2018.

[8] D.P. Bertsekas. Reinforcement Learning and Optimal Control. Athena Scientific, 2019.

[9] Ronen Brafman, Giuseppe De Giacomo, and Fabio Patrizi. LTLf/LDLf Non-Markovian Rewards. Proceedings of the AAAI Conference on Artificial Intelligence, 32(1), April 2018. Number: 1.

[10] Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning. In Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, pages 6065–6073, Macao, China, August 2019.

[11] Christos G Cassandras and Stéphane Lafortune. Introduction to Discrete Event Systems. Springer, 2008.

[12] Thomas Cederborg, Ishaan Grover, Charles L. Isbell, and Andrea L. Thomaz. Policy Shaping with Human Teachers. In Twenty-Fourth International Joint Conference on Artificial Intelligence, June 2015.

[13] Richard Cheng, Gábor Orosz, Richard M Murray, and Joel 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, Honolulu, Hawaii, USA, 2019.

[14] Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. arXiv preprint arXiv:1801.08757, 2018.

[15] Floris den Hengst, Vincent Franois-Lavet, Mark Hoogendoorn, and Frank van Harmelen. Planning for potential: efficient safe reinforcement learning. Machine Learning, 111(6):2255–2274, June 2022.

[16] Javier García and Fernando Fernández. Safe exploration of state and action spaces in reinforcement learning. Journal of Artificial Intelligence Research, 45:515–564, 2012.

[17] Javier García and Fernando Fernández. A comprehensive survey on safe reinforcement learning. Journal of Machine Learning Research, 16(1): 1437–1480, 2015.

[18] Vijay K Garg, Ratnesh Kumar, and Steven I Marcus. A probabilistic language formalism for stochastic discrete-event systems. IEEE Transactions on Automatic Control, 44(2):280–293, 1999.

[19] Chris Gaskett. Reinforcement learning under circumstances beyond its control. In Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation, Vienna, Austria, 2003.

[20] Clement Gehring and Doina Precup. Smart exploration in reinforcement learning using absolute temporal difference errors. In Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems, Saint Paul, USA, 2013.

[21] P Geibel and F Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. Journal of Artificial Intelligence Research, 24:81–108, 2005.

[22] Peter Geibel. Reinforcement learning for MDPs with constraints. In Proceedings of the 17th European Conference on Machine Learning, Berlin, Heidelberg, 2006.

[23] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. Restraining Bolts for Reinforcement Learning Agents. Proceedings of the AAAI Conference on Artificial Intelligence, 34:13659–13662, April 2020.

[24] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles L Isbell, and Andrea L Thomaz. Policy Shaping: Integrating Human Feedback with Reinforcement Learning. In Advances in Neural Information Processing Systems, volume 26. Curran Associates, Inc., 2013.

[25] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-Constrained Reinforcement Learning, February 2019. arXiv:1801.08099.

[26] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Cautious Reinforcement Learning with Logical Constraints. In Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '20, pages 483–491, Richland, SC, May 2020.

[27] Matthias Heger. Consideration of risk in reinforcement learning. In Machine Learning Proceedings 1994, pages 105–111. Elsevier, 1994.

[28] John E Hopcroft and Jeffrey D Ullman. Introduction to Automata Theory, Languages and Computation. Adison-Wesley, 1979.

[29] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning. In Proceedings of the 35th International Conference on Machine Learning, pages 2107–2116, July 2018.

[30] Bettina Knighofer, Julian Rudolf, Alexander Palmisano, Martin Tappler, and Roderick Bloem. Online shielding for reinforcement learning. Innovations in Systems and Software Engineering, September 2022.

[31] Bettina Könighofer, Mohammed Alshiekh, Roderick Bloem, Laura Humphrey, Robert Könighofer, Ufuk Topcu, and Chao Wang. Shield synthesis. Formal Methods in System Design, 51(2):332–361, 2017.

[32] R. Kumar and V.K. Garg. Control of stochastic discrete event systems modeled by probabilistic languages. IEEE Transactions on Automatic Control, 46(4):593–606, April 2001.

[33] Adam Laud and Gerald DeJong. The influence of reward on the speed of reinforcement learning: an analysis of shaping. In Proceedings of the Twentieth International Conference on International Conference on Machine Learning, ICML'03, pages 440–447, Washington, DC, USA, August 2003.

[34] M Lawford and Walter Murray Wonham. Supervisory control of probabilistic discrete event systems. In Proceedings of 36th Midwest Symposium on Circuits and Systems, Detroit, MI, USA, 1993.

[35] Xiao Li, Cristian-Ioan Vasile, and Calin Belta. Reinforcement learning with temporal logic rewards. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 3834–3839, September 2017.

[36] Michael L. Littman, Ufuk Topcu, Jie Fu, Charles Isbell, Min Wen, and James MacGlashan. Environment-Independent Task Specifications via GLTL, April 2017. arXiv:1704.04341.

[37] Travis Mandel, Yun-En Liu, Emma Brunskill, and Zoran Popovi. Where to Add Actions in Human-in-the-Loop Reinforcement Learning. Proceedings of the AAAI Conference on Artificial Intelligence, 31(1), February 2017.

[38] Kunal Menda, K. Driggs-Campbell, and Mykel J. Kochenderfer. Ensembledagger: A bayesian approach to safe imitation learning. In Proceedings of the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, Macao, 2019.

[39] Oliver Mihatsch and Ralph Neuneier. Risk-sensitive reinforcement learning. Machine Learning, 49(2):267–290, 2002.

[40] Teodor Mihai Moldovan and Pieter Abbeel. Safe exploration in markov decision processes. In Proceedings of the 29th International Coference on International Conference on Machine Learning, Madison, WI, USA, 2012.

[41] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming Exploration in Reinforcement Learning with Demonstrations. In 2018 IEEE International Conference on Robotics and Automation (ICRA), pages 6292–6299, May 2018.

[42] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In Proceedings of the Sixteenth International Conference on Machine Learning, ICML '99, pages 278–287, San Francisco, CA, USA, June 1999.

[43] Vera Pantelic, Steven M Postma, and Mark Lawford. Probabilistic supervisory control of probabilistic discrete event systems. IEEE Transactions on Automatic Control, 54(8):2013–2018, 2009.

[44] Martin Pecka and Tomas Svoboda. Safe exploration techniques for reinforcement learning–an overview. In Proceedings of the First International Wokshop on International Workshop on Modelling and Simulation for Autonomous Systems, Rome, Italy, 2014.

[45] Karl Pertsch, Youngwoon Lee, and Joseph Lim. Accelerating Reinforcement Learning with Learned Skill Priors. In Proceedings of the 2020 Conference on Robot Learning, pages 188–204, October 2021.

[46] Peter J Ramadge and W Murray Wonham. Supervisory control of a class of discrete event processes. SIAM Journal on Control and Optimization, 25(1):206–230, 1987.

[47] Herbert Robbins and Sutton Monro. A stochastic approximation method. The Annals of Mathematical Statistics, 22(3):400–407, 1951.

[48] Benjamin Rosman and Subramanian Ramamoorthy. What good are actions? Accelerating learning using learned action priors. In 2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL), pages 1–6, 2012.

[49] Stephane Ross and Drew Bagnell. Efficient reductions for imitation learning. In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, Sardinia, Italy, 2010.

[50] Alexander A. Sherstov and Peter Stone. Improving action selection in MDP's via knowledge transfer. In Proceedings of the 20th national conference on Artificial intelligence - Volume 2, AAAI'05, pages 1024–1029, Pittsburgh, Pennsylvania, July 2005.

[51] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. MIT Press, 2018.

[52] Martin Tappler, Stefan Pranger, Bettina Knighofer, Edi Mukardin, Roderick Bloem, and Kim Larsen. Automata Learning Meets Shielding. In Tiziana Margaria and Bernhard Steffen, editors, Leveraging Applications of Formal Methods, Verification and Validation. Verification Principles, Lecture Notes in Computer Science, pages 335–359. Springer International Publishing, 2022.

[53] Lisa Torrey and Matthew Taylor. Teaching on a budget: Agents advising agents in reinforcement learning. In Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems, Saint Paul, USA, 2013.

[54] W Murray Wonham and Kai Cai. Supervisory Control of Discrete-Event Systems. Springer, 2019.

[55] Jiakai Zhang and Kyunghyun Cho. Query-efficient imitation learning for end-to-end simulated driving. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 2017.

[56] Matthieu Zimmer, Paolo Viappiani, and Paul Weng. Teacher-student framework: a reinforcement learning approach. In AAMAS Workshop Autonomous Robots and Multirobot Systems, 2014.