特　別　研　究　報　告

題　目

Reinforcement Learning based Controller synthesis
for Linear Temporal Logic Specifications
Using Limit-Deterministic Generalized Büchi Automata

指 導 教 員

Professor Toshimitsu Ushio

報 告 者

Ryohei Oura

February 17, 2020

大阪大学基礎工学部システム科学科
知能システム学コース

Abstract

In this thesis, we propose a novel reinforcement learning method for the synthesis of a controller satisfying a control specification described by a linear temporal logic formula and apply the proposed method to supervisory control. We assume that the controlled system is modeled by a Markov decision process (MDP). We transform the specification to a limit-deterministic generalized Büchi automaton (LDGBA) with several accepting sets that accepts all infinite sequences satisfying the formula. The LDGBA is augmented so that it explicitly records the previous visits to accepting sets. We take a product of the augmented LDGBA and the MDP, based on which we define a reward function. The agent gets rewards whenever state transitions are in an accepting set that has not been visited for a certain number of steps and the value function is maximized when all accepting sets are visited infinitely often. Consequently, sparsity of rewards is relaxed and optimal circulations among the accepting sets are learned. We show that the proposed method can learn an optimal policy when the discount factor is sufficiently close to one.

Contents

Abstract **i**

1 Introduction **1**

2 Preliminaries **3**
- 2.1 Markov Decision Processes . . . 3
- 2.2 Reinforcement Learning . . . 4
  - 2.2.1 Objective functions and an Optimal policy . . . 4
  - 2.2.2 Temporal Difference Learning . . . 7
- 2.3 Linear Temporal Logic and Automata . . . 8

3 Reinforcement learning based control policy synthesis for LTL specifications **10**
- 3.1 Augmentation of tLDGBAs and Synthesis Method . . . 10
- 3.2 Example . . . 13

4 Reinforcement learning based supervisor synthesis for LTL specifications **17**
- 4.1 Stochastic Discrete Event Systems . . . 17
- 4.2 Product DESs . . . 19
- 4.3 Learning Algorithm . . . 20
- 4.4 Example . . . 22

5 Conclusions **26**

A Proofs **27**

Acknowledgment **31**

References **32**

Chapter 1

Introduction

Temporal logic has been developed in computer engineering as a useful formalism of formal specifications [1, 2]. A merit of temporal logics is its resemblance to natural languages and it has been widely used in several other areas of engineering. Especially, a complicated mission or task in computer-controlled systems such as robots can be described by a temporal logic specification precisely and many synthesis algorithms of a controller or a planner that satisfies the specification have been proposed [3, 4, 5, 6]. Linear temporal logic (LTL) is often used as a specification language because of its rich expressiveness. It can explain many important ω-regular properties such as liveness, safety, and persistence [1]. It is known that the LTL specification is converted into an ω-automaton such as a nondeterministic Büchi automaton and a deterministic Rabin automaton [1, 7]. In the synthesis of a control policy for the LTL specification, we model a controlled system by a transition system that abstracts its dynamics, construct a product automaton of the transition system and the ω-automaton corresponding to the LTL specification, and compute a winning strategy of a game over the product automaton [7].

Because of inherent stochasticity of controlled systems, we often use a Markov decision process (MDP) as a finite-state abstraction of the controlled systems [8]. In the case where the probabilities are unknown a priori, we have two approaches to the synthesis of the control policy. One is robust control where we assume that state transition probabilities are in uncertainty sets [9] while the other is learning using samples [10].

Reinforcement learning (RL) is a useful approach to learning an optimal policy from sample behaviors of the controlled system [11]. In RL, we use a reward function that assigns a reward to each transition in the behaviors and evaluate a control policy by the return that is an expected (discounted) sum of the rewards along the behaviors. Thus, to apply RL to the synthesis of a control policy for the LTL specification, it is an important issue how to introduce the reward function, which depends on the acceptance condition of an ω-automaton converted from the LTL specification. A reward function based on the acceptance condition of a Rabin automaton was proposed in [10]. It was applied to a control problem where the controller optimizes a given control cost under the LTL constraint [12].

Recently, a limit-deterministic Büchi automaton (LDBA) or generalized one (LDGBA) is paid much attention to as an ω-automaton corresponding to the LTL specification [13]. The RL-based approaches to the synthesis of a control policy using LDBAs have been proposed in [14, 15, 16, 17]. In [15, 17], they use an LDBA. However, when constructing a Büchi automaton (BA) from a generalized Büchi automaton (GBA), the order of visits to accepting sets of the BA is fixed. The construction causes the sparsity of the reward based on the acceptance condition of a BA. On the other hand, to deal with the acceptance condition of an LDGBA that accepts behaviors visiting all accepting sets infinitely often, the accepting frontier function was introduced in [14, 16]. The reward function is defined based on the function. However, the function is memoryless, that is, it does not provide the information of accepting sets that have been visited, which is important to improve learning performance. In this letter, we propose a novel method to augment an LDGBA converted from a given LTL formula. Then, we define a reward function based on the acceptance condition of the product MDP of the augmented LDGBA and the controlled system. As a result, we can improve the sparsity of rewards and expand the class of policies that satisfy the LTL specification compared to [14].

The rest of this thesis is organized as follows. Chapter 2 reviews Markov decision processes, reinforcement learning, linear temporal logic, and automata. Chapter 3 proposes a novel reinforcement learning based method for the synthesis of control policies. Chapter 4 proposes a novel reinforcement learning based supervisor synthesis using the method introduced in Chapter 3. Chapter 5 concludes the results and future works.

Chapter 2

Preliminaries

Notations For sets A and B, AB denotes their concatenation. A^ω denotes the infinite concatenation of the set A and A^* denotes the finite one. $\mathbb{N}_0$ is the set of nonnegative integers. $\mathbb{R}_{\geq 0}$ is the set of nonnegative real numbers.

2.1 Markov Decision Processes

We define a controlled system as a labeled Markov decision process.

Definition 2.1 (Labeled Markov Decision Process) A (labeled) Markov decision process (MDP) is a tuple $M = (S, A, P, s_{init}, AP, L)$, where S is a finite set of states, A is a finite set of actions, $P : S \times S \times A \to [0, 1]$ is a transition probability function, $s_{init} \in S$ is the initial state, AP is a finite set of atomic propositions, and $L : S \times A \times S \to 2^{AP}$ is a labeling function that assigns a set of atomic propositions to each transition. Let $\mathcal{A}(s) = \{a \in A; \exists s' \in S \text{ s.t. } P(s'|s, a) \neq 0\}$. Note that $\sum_{s' \in S} P(s'|s, a) = 1$ holds for any state $s \in S$ and action $a \in \mathcal{A}(s)$.

In the MDP M, an infinite path starting from a state $s_0 \in S$ is defined as a sequence $\rho = s_0 a_0 s_1 \ldots \in S(AS)^\omega$ such that $P(s_{i+1}|s_i, a_i) > 0$ for any $i \in \mathbb{N}_0$. A finite path is a finite sequence in $S(AS)^*$. In addition, we sometimes represent ρ as ρ_{init} to emphasize that ρ starts from $s_0 = s_{init}$. For a path $\rho = s_0 a_0 s_1 \ldots$, we define the corresponding labeled path $L(\rho) = L(s_0, a_0, s_1) L(s_1, a_1, s_2) \ldots \in (2^{AP})^\omega$. $InfPath^M$ (resp., $FinPath^M$) is defined as the set of infinite (resp., finite) paths starting from $s_0 = s_{init}$ in the MDP M. For each finite path ρ, $last(\rho)$ denotes its last state.

Definition 2.2 (Policy) A policy on an MDP M is defined as a mapping $\pi : FinPath^M \times \mathcal{A}(last(\rho)) \to [0, 1]$. A policy π is a *positional* policy if for any $\rho \in FinPath^M$ and any $a \in \mathcal{A}(last(\rho))$, it holds that $\pi(\rho, a) = \pi(last(\rho), a)$ and there exists $a' \in \mathcal{A}(last(\rho))$ such that

$$\pi(\rho, a) = \begin{cases} 1 & \text{if } a = a', \\ 0 & \text{otherwise.} \end{cases}$$

Let $InfPath_{\pi}^{M}$ (resp., $FinPath_{\pi}^{M}$) be the set of infinite (resp., finite) paths starting from $s_0 = s_{init}$ in the MDP M under a policy π. The behavior of an MDP M under a policy π is defined on a probability space $(InfPath_{\pi}^{M}, \mathcal{F}_{InfPath_{\pi}^{M}}, Pr_{\pi}^{M})$.

Definition 2.3 (Markov chain) A Markov chain induced by an MDP M with a positional policy π is a tuple $MC_{\pi} = (S_{\pi}, P_{\pi}, s_0, AP, L)$, where $S_{\pi} = S$, $P_{\pi}(s'|s) = P(s'|s,a)$ for $s, s' \in S$ and $a \in \mathcal{A}(s)$ such that $\pi(s,a) = 1$. The state set S_{π} of MC_{π} can be represented as a disjoint union of a set of transient states T_{π} and closed irreducible sets of recurrent states R_{π}^{j} with $j \in \{1, \ldots, h\}$, i.e., $S_{\pi} = T_{\pi} \cup R_{\pi}^{1} \cup \ldots \cup R_{\pi}^{h}$ [18]. In the following, we say a “recurrent class” instead of a “closed irreducible set of recurrent states” for simplicity.

In an MDP M, we define a reward function $\mathcal{R} : S \times A \times S \to \mathbb{R}$, where $\mathbb{R}$ is the set of real numbers. The function denotes the immediate scalar bounded reward received after the agent performs an action a at a state s and reaches a next state s' as a result.

2.2 Reinforcement Learning

Reinforcement learning is the theoretical framework to find a policy maximizing or minimizing an objective function through the iterative interactions between the learner referred to the agent and the controlled system referred to the environment. The interaction is that the agent takes an action on the environment and the environment returns an observation such as an immediate reward or a next state. In this section, since we use model-free reinforcement learning methods in this thesis, we introduce the model-free reinforcement learning methods, which find a policy maximizing or minimizing the objective function without explicit estimations of the environment.

2.2.1 Objective functions and an Optimal policy

Definition 2.4 (Expected discounted reward) For a policy π on an MDP M, any state $s \in S$, and a reward function $\mathcal{R}$, we define the expected discounted reward as

$$V^{\pi}(s) = \mathbb{E}^{\pi}[\sum_{n=0}^{\infty} \gamma^{n} \mathcal{R}(S_n, A_n, S_{n+1}) | S_0 = s],$$

where $\mathbb{E}^{\pi}$ denotes the expected value given that the agent follows the policy π from the state s and $\gamma \in [0, 1)$ is a discount factor. Intuitively, the magnitude of the discount factor

γ determines how much we consider rewards received in the future. The closer γ is to 0, the more important the rewards of near future are. On the other hand, the rewards of far future are also important when γ is closer to 1. The function $V^\pi(s)$ is often referred to as a state-value function under the policy π. For any state-action pair $(s, a) \in S \times A$, we define an action-value function $Q^\pi(s, a)$ under the policy π as follows.

$$Q^\pi(s,a) = \mathbb{E}^\pi[\sum_{n=0}^{\infty} \gamma^n \mathcal{R}(S_n, A_n, S_{n+1}) | S_0 = s, A_0 = a].$$

We have the following recursively equation for the state-value function and the action-value function.

$$\begin{aligned}
V^\pi(s) =& \mathbb{E}^\pi[\sum_{n=0}^{\infty} \gamma^n \mathcal{R}(S_n, A_n, S_{n+1}) | S_0 = s] \\
=& \sum_{a \in \mathcal{A}(s)} \pi(s,a) \sum_{s' \in S} P(s'|s,a) \mathbb{E}^\pi[\sum_{n=0}^{\infty} \gamma^n \mathcal{R}(S_n, A_n, S_{n+1}) | S_0 = s, A_0 = a, S_1 = s'] \\
=& \sum_{a \in \mathcal{A}(s)} \pi(s,a) \sum_{s' \in S} P(s'|s,a) \{\mathcal{R}(s,a,s') + \gamma \mathbb{E}^\pi[\sum_{n=0}^{\infty} \gamma^n \mathcal{R}(S_n, A_n, S_{n+1}) | S_1 = s']\} \\
=& \sum_{a \in \mathcal{A}(s)} \pi(s,a) \sum_{s' \in S} P(s'|s,a) \{\mathcal{R}(s,a,s') + \gamma V^\pi(s')\}, \qquad (2.1)
\end{aligned}$$

by the definition of the action-value function, we have

$$\begin{aligned}
V^\pi(s) &= \sum_{a \in \mathcal{A}(s)} \pi(s,a) Q^\pi(s,a), \\
Q^\pi(s,a) &= \sum_{s' \in S} P(s'|s,a) \{\mathcal{R}(s,a,s') + \gamma V^\pi(s')\} \\
&= \sum_{s' \in S} P(s'|s,a) \{\mathcal{R}(s,a,s') + \gamma \sum_{a' \in \mathcal{A}(s')} \pi(s',a') Q^\pi(s',a')\}. \qquad (2.2)
\end{aligned}$$

The above equations are called the *Bellman equations*.

Definition 2.5 (Optimal policy) For any state $s \in S$, a policy π^* is optimal if

$$\pi^* \in \arg\max_{\pi \in \Pi^{pos}} V^\pi(s),$$

where Π^{pos} is the set of positional policies over the state set S.

We have the following *Bellman optimality functions* by the definition of optimal policies.

$$
\begin{aligned}
V^*(s) :=& V^{\pi^*}(s) \\
=& \max_{\pi \in \Pi^{pos}} V^\pi(s) \\
=& \max_{\pi \in \Pi^{pos}} \sum_{a \in \mathcal{A}(s)} \pi(s,a) \sum_{s' \in S} P(s'|s,a)\{\mathcal{R}(s,a,s') + \gamma V^\pi(s')\} \\
=& \max_{a \in \mathcal{A}(s)} [\sum_{s' \in S} P(s'|s,a)\{\mathcal{R}(s,a,s') + \gamma V^{\pi^*}(s')\}],
\end{aligned} \tag{2.3}
$$

$$
\begin{aligned}
Q^*(s,a) :=& Q^{\pi^*}(s,a) \\
=& \max_{\pi \in \Pi^{pos}} Q^\pi(s,a) \\
=& \max_{\pi \in \Pi^{pos}} \sum_{s' \in S} P(s'|s,a)\{\mathcal{R}(s,a,s') + \gamma \sum_{a' \in \mathcal{A}(s')} \pi(s',a') Q^\pi(s',a')\} \\
=& \sum_{s' \in S} P(s'|s,a)\{\mathcal{R}(s,a,s') + \gamma \max_{\pi \in \Pi^{pos}} \sum_{a' \in \mathcal{A}(s')} \pi(s',a') Q^\pi(s',a')\} \\
=& \sum_{s' \in S} P(s'|s,a)\{\mathcal{R}(s,a,s') + \gamma \max_{a \in \mathcal{A}(s')} Q^\pi(s',a')\}.
\end{aligned} \tag{2.4}
$$

We call V^* and Q^* the optimal state-value function and the optimal action-value function, respectively. $V^*(s)$ represents $Q^*(s,a)$ with an optimal action at the first step. Therefore, for any state $s \in S$, we have

$$
V^*(s) = \max_{a \in \mathcal{A}(s)} Q^*(s,a).
$$

In words, the set of optimal policies under V^* and the set of optimal policies under Q^* are the same.

If we know the full and accurate information of an MDP such as the transition probability or the reward function, we can obtain an optimal policy by solving Eqs. (2.3) or (2.4) directly. We usually use *Dynamic Programming* by solving recursively Eq. (2.3) or Eq. (2.4). To find V^π or Q^π for a policy π by solving Eq. (2.1) or Eq. (2.2) is referred to *Policy Evaluation*. For any state $s \in S$ or any state-action pair $(s,a) \in S \times A$, the update of the policy π in order to increase the value of $V^\pi(s)$ or $Q^\pi(s,a)$ is referred to *Policy Improvement*. The method that finds an optimal policy by updating the optimal value function repeatedly in accordance with Eqs. (2.3) or (2.4) is referred to *Value Iteration*. The method that finds an optimal policy by repeating policy evaluation and policy improvement alternately is referred to *Policy Iteration*.

2.2.2 Temporal Difference Learning

If the MDP M is unknown, we can not use Dynamic programming such as value iteration or policy iteration to obtain an optimal policy. When the MDP M is unknown, we often use reinforcement learning to find an optimal policy instead of dynamic programming.

Temporal difference learning (TD-learning) is the basic method of model-free reinforcement learning. The method does not require the prior knowledge about the environment and utilizes a raw experience by one step. Unlike dynamic programing, we update a value function using the experience in an on-line manner as follows.

$$\hat{V}^{\pi_k}(s_k) \leftarrow \hat{V}^{\pi_k}(s_k) + \alpha_k\{r_{k+1} + \hat{V}^{\pi_k}(s_{k+1}) - \hat{V}^{\pi_k}(s_k)\}, \tag{2.5}$$

where π_k, s_k, and α_k are the policy, the state, and the learning ratio at the time step k, respectively, and $r_{k+1} = \mathcal{R}(s_k, a_k, s_{k+1})$. Note that $\alpha_k \in [0,1]$ for any $k \in \mathbb{N}_0$. The quantity in the curly bracket in the right hand side is called a *TD-error*:

$$\Delta_k = r_{k+1} + \hat{V}^{\pi_k}(s_{k+1}) - \hat{V}^{\pi_k}(s_k). \tag{2.6}$$

The TD-error at time step k represents the difference between the current estimated value of s_k and the better estimated value of s_k based on the actual experience, namely $r_{k+1} + \hat{V}^{\pi_k}(s_{k+1})$. Intuitively, each time the value function is updated, the TD errors are gradually reduced and the estimated value function converges to the true value function under the policy π if π_k goes to a policy π. Hence, the magnitude of the learning ratio α_k describes how much we influence the most recent experience on updating the current estimated value.

TD-learning methods for an action-value function are classified into two main learning methods *Q-learning* and *SARSA*. In Q-learning, we do not use the actual action at the next state to update the current estimated state-action value of the current state-action pair. Instead, we use the optimal action at the next state to update the current estimated state-action value of the current state-action pair. Thus, the way of update the estimated state-action value $\hat{Q}^{\pi_k}(s_k, a_k)$ at time step k is as follows.

$$\hat{Q}^{\pi_k}(s_k, a_k) \leftarrow \hat{Q}^{\pi_k}(s_k, a_k) + \alpha_k\{r_{k+1} + \max_{a' \in \mathcal{A}(s_{k+1})} \hat{Q}^{\pi_k}(s_{k+1}, a') - \hat{Q}^{\pi_k}(s_k, a_k)\}. \tag{2.7}$$

On the other hand, in SARSA, we use the actual action at the next state to update the current estimated state-action value of the current state-action pair. That is,

$$\hat{Q}^{\pi_k}(s_k, a_k) \leftarrow \hat{Q}^{\pi_k}(s_k, a_k) + \alpha_k\{r_{k+1} + \hat{Q}^{\pi_k}(s_{k+1}, a_{k+1}) - \hat{Q}^{\pi_k}(s_k, a_k)\}. \tag{2.8}$$

2.3 Linear Temporal Logic and Automata

In our proposed method, we use linear temporal logic (LTL) formulas to describe various constraints or properties and to systematically assign corresponding rewards. LTL formulas are constructed from a set of atomic propositions, Boolean operators, and temporal operators. We use the standard notations for the Boolean operators: $\top$ (true), $\neg$ (negation), and $\wedge$ (conjunction). LTL formulas over a set of atomic propositions AP are recursively defined as

$$\varphi ::= \top \mid \alpha \in AP \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \mathbf{X}\varphi \mid \varphi_1 \mathbf{U} \varphi_2,$$

where φ, φ_1, and φ_2 are LTL formulas. Additional Boolean operators are defined as $\perp := \neg\top$, $\varphi_1 \vee \varphi_2 := \neg(\neg\varphi_1 \wedge \neg\varphi)$, and $\varphi_1 \Rightarrow \varphi_2 := \neg\varphi_1 \vee \varphi_2$. The operators $\mathbf{X}$ and $\mathbf{U}$ are called "next" and "until", respectively. Using the operator $\mathbf{U}$, we define two temporal operators: (1) *eventually*, $\mathbf{F}\varphi := \top\mathbf{U}\varphi$ and (2) *always*, $\mathbf{G}\varphi := \neg\mathbf{F}\neg\varphi$.

Let M be an MDP. For an infinite path $\rho = s_0 a_0 s_1 \ldots$ of M with $s_0 \in S$, let $\rho[i]$ be the i-th state of ρ i.e., $\rho[i] = s_i$ and let $\rho[i:]$ be the i-th suffix $\rho[i:] = s_i a_i s_{i+1} \ldots$. We define the i-th state and i-th suffix of the infinite path for a DES in the same way.

Definition 2.6 (LTL semantics) For an LTL formula φ, an MDP M, and an infinite path $\rho = s_0 a_0 s_1 \ldots$ of M with $s_0 \in S$, the satisfaction relation $M, \rho \models \varphi$ is recursively defined as follows.

$$\begin{aligned}
&M, \rho \models \top, \\
&M, \rho \models \alpha \in AP \Leftrightarrow \alpha \in L(s_0, a_0, s_1), \\
&M, \rho \models \varphi_1 \wedge \varphi_2 \Leftrightarrow M, \rho \models \varphi_1 \wedge M, \rho \models \varphi_2, \\
&M, \rho \models \neg\varphi \qquad \Leftrightarrow M, \rho \not\models \varphi, \\
&M, \rho \models \mathbf{X}\varphi \qquad \Leftrightarrow M, \rho[1:] \models \varphi, \\
&M, \rho \models \varphi_1 \mathbf{U} \varphi_2 \ \Leftrightarrow \exists j \geq 0,\ M, \rho[j:] \models \varphi_2 \wedge \forall i, 0 \leq i < j,\ M, \rho[i:] \models \varphi_1.
\end{aligned}$$

The next operator $\mathbf{X}$ requires that φ is satisfied by the next state suffix of ρ. The until operator $\mathbf{U}$ requires that φ_1 holds true until φ_2 becomes true over the path ρ. For the path in a DES, we define the LTL semantics in the same way. Using the operator $\mathbf{U}$ we can define two temporal operators: (1) *eventually*, $\mathbf{F}\varphi := \top\mathbf{U}\varphi$ and (2) *always*, $\mathbf{G}\varphi := \neg\mathbf{F}\neg\varphi$. In the following, we write $\rho \models \varphi$ for simplicity without referring to MDP M and DES D.

For any policy π, the probability of all paths starting from s_{init} on the MDP M that satisfy an LTL formula φ under the policy π, or the satisfaction probability is defined as follows.

$$Pr_\pi^M(s_{init} \models \varphi) := Pr_\pi^M(\{\rho_{init} \in InfPath_\pi^M; \rho_{init} \models \varphi\}).$$

We say that an LTL formula φ is satisfied by a positional policy π (resp., a supervisor SV) if

$$Pr_{\pi}^{M}(s_{init} \models \varphi) > 0.$$

Any LTL formula φ can be converted into various automata, namely finite state machines that recognize all words satisfying φ. We define a generalized Büchi automaton at the beginning, and then introduce a limit-deterministic Büchi automaton [16].

Definition 2.7 (Transition-based generalized Büchi automata) A transition-based generalized Büchi automaton (tGBA) is a tuple $B = (X,\ x_{init},\ \Sigma,\ \delta,\ \mathcal{F})$, where X is a finite set of states, $x_{init} \in X$ is the initial state, Σ is an input alphabet including ε, $\delta \subset X \times \Sigma \times X$ is a set of transitions, and $\mathcal{F} = \{F_1, \ldots, F_n\}$ is an acceptance condition, where for each $j \in \{1, \ldots, n\}$, $F_j \subset \delta$ is a set of accepting transitions and called an accepting set. We refer to a tGBA with one accepting set as a tBA.

Let Σ^{ω} be the set of all infinite words over Σ and let an infinite run be an infinite sequence $r = x_0\sigma_0x_1 \ldots \in X(\Sigma X)^{\omega}$ where $(x_i, \sigma_i, x_{i+1}) \in \delta$ for any $i \in \mathbb{N}_0$. An infinite word $w = \sigma_0\sigma_1 \ldots \in \Sigma^{\omega}$ is accepted by B_{φ} if and only if there exists an infinite run $r = x_0\sigma_0x_1 \ldots$ starting from $x_0 = x_{init}$ such that $inf(r) \cap F_j \neq \emptyset$ for each $F_j \in \mathcal{F}$, where $inf(r)$ is the set of transitions that occur infinitely often in the run r.

Definition 2.8 (Sink state) A sink state in state set X of a tLDBA $B = (X, x_{init}, \Sigma, \delta, \mathcal{F})$ is defined as a state such that there exist no accepting transition of B that is accessible from the state. We denote the set of sink states as $SinkSet$.

Definition 2.9 (Limit-deterministic generalized Büchi automata) A transition-based limit-deterministic generalized Büchi automaton (tLDGBA) is a tGBA $B = (X, x_{init}, \Sigma, \delta, \mathcal{F})$ such that X is partitioned into two disjoint sets $X_{initial}$ and X_{final} such that

- $F_j \subset X_{final} \times \Sigma \times X_{final}$, $\forall j \in \{1, ..., n\}$,
- $|\{(x, \sigma, x') \in \delta; \sigma \in \Sigma, x' \in X_{final}\}| = 1$, $\forall x \in X_{final}$,
- $|\{(x, \sigma, x') \in \delta; \sigma \in \Sigma, x' \in X_{initial}\}| = 0$, $\forall x \in X_{final}$,
- $\forall (x, \varepsilon, x') \in \delta,\ x \in X_{initial} \wedge x' \in X_{final}$.

An ε-transition enables the tLDGBA to change its state with no input. Then, ε-transitions reflect a single "guess" from $X_{initial}$ to X_{final}. Note that by the construction in [13], the transitions in each part are deterministic except for ε-transitions from X_{intial} to X_{final}. It is known that, for any LTL formula φ, there exists a tLDGBA that accepts all words satisfying φ [13]. We refer to a tLDGBA with one accepting set as a tLDBA. In particular, we denote a tLDGBA recognizing an LTL formula φ by B_{φ}, whose input alphabet is given by $\Sigma = 2^{AP} \cup \{\varepsilon\}$.

Chapter 3

Reinforcement learning based control policy synthesis for LTL specifications

3.1 Augmentation of tLDGBAs and Synthesis Method

We introduce an automaton augmented with binary vectors. The automaton can explicitly represent whether transitions in each accepting set occur at least once, and ensure transitions in each accepting set occur infinitely often.

Let $V = \{(v_1, \ldots, v_n)^T \ ; \ v_i \in \{0, 1\}, \ i \in \{1, \ldots, n\}\}$ be a set of binary-valued vectors, and let $\mathbf{1}$ and $\mathbf{0}$ be the n-dimentional vectors with all elements 1 and 0, respectively. In order to augment a tLDBA B_φ, we introduce three functions $visitf : \delta \to V$, $reset : V \to V$, and $Max : V \times V \to V$ as follows. For any $e \in \delta$, $visitf(e) = (v_1, \ldots, v_n)^T$, where

$$v_i = \begin{cases} 1 & \text{if } e \in F_i, \\ 0 & \text{otherwise.} \end{cases}$$

For any $v \in V$,

$$reset(v) = \begin{cases} \mathbf{0} & \text{if } v = \mathbf{1}, \\ v & \text{otherwise.} \end{cases}$$

For any $v, u \in V$, $Max(v, u) = (l_1, \ldots, l_n)^T$, where $l_i = max\{v_i, u_i\}$ for any $i \in \{1, \ldots, n\}$.

Each vector v is called a memory vector and represents which accepting sets have been visited. The function $visitf$ returns a binary vector whose i-th element is 1 if and only if a transition in the accepting set F_i occurs. The function $reset$ returns the zero vector $\mathbf{0}$ if at least one transition in each accepting set has occurred after the latest reset. Otherwise, it returns the input vector without change.

Definition 3.1 (Augmented Automata) For a tLDGBA $B_\varphi = (X, x_{init}, \Sigma, \delta, \mathcal{F})$, its augmented automaton is a tLDGBA $\bar{B}_\varphi = (\bar{X}, \bar{x}_{init}, \bar{\Sigma}, \bar{\delta}, \bar{\mathcal{F}})$, where $\bar{X} = X \times V$, $\bar{x}_{init} = (x_{init}, \mathbf{0})$, $\bar{\Sigma} = \Sigma$, $\bar{\delta}$ is defined as $\bar{\delta} = \{((x, v), \bar{\sigma}, (x', v')) \in \bar{X} \times \bar{\Sigma} \times \bar{X}\ ;\ (x, \bar{\sigma}, x') \in \delta,\ v' = reset(Max(v, visitf((x, \bar{\sigma}, x'))))\}$, and $\bar{\mathcal{F}} = \{\bar{F}_1, \ldots, \bar{F}_n\}$ is defined as $\bar{F}_i = \{((x, v), \bar{\sigma}, (x', v')) \in \bar{\delta}\ ;\ (x, \sigma, x') \in F_i,\ v_i = 0\}$ for each $i \in \{1, ..., n\}$.

We denote by $\mathcal{L}(B)$ the accepted language of a tLDGBA B, namely the set of all infinite words accepted by B.

Proposition 3.1 Let $B = (X, x_{init}, \Sigma, \delta, \mathcal{F})$ and $\bar{B} = (\bar{X}, \bar{x}_{init}, \bar{\Sigma}, \bar{\delta}, \bar{\mathcal{F}})$ be an arbitrary tLDGBA and its augmentation, respectively. Then, we have $\mathcal{L}(B) = \mathcal{L}(\bar{B})$.

The proof of Proopsition 3.1 is shown in Appendix A.

The augmented tLDGBA $\bar{B}_\varphi$ keeps track of previous visits to the accepting sets of B_φ. Intuitively, along a run of $\bar{B}_\varphi$, a memory vector v is reset to $\mathbf{0}$ when at least one transition in each accepting set of the original tLDGBA B_φ has occurred.

For example, shown in Figs. 3.1 and 3.2 are a tLDGBA and its augmented automaton, respectively, associated with the following LTL formula.

$$\varphi = \mathbf{GF}a \wedge \mathbf{GF}b \wedge \mathbf{G}\neg c. \tag{3.1}$$

The acceptance condition $\mathcal{F}$ of the tLDGBA is given by $\mathcal{F} = \{F_1, F_2\}$, where $F_1 = \{(x_0, \{a\}, x_0),\ (x_0, \{a, b\}, x_0)\}$ and $F_2 = \{(x_0, \{b\}, x_0),\ (x_0, \{a, b\}, x_0)\}$. Practically, states in a strongly connected component that contains no accepting transitions can be merged as shown in Fig. 3.2.

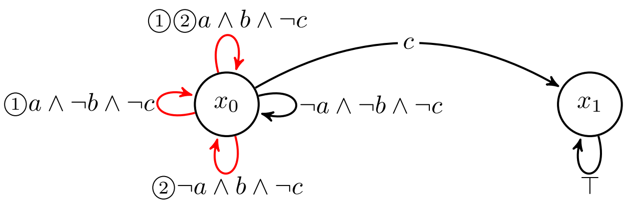

Fig. 3.1 The tLDGBA recognizing the LTL formula $\mathbf{GF}a \wedge \mathbf{GF}b \wedge \mathbf{G}\neg c$, where the initial state is x_0. Red arcs are accepting transitions that are numbered in accordance with the accepting sets they belong to.

We modify the standard definition of a product MDP to deal with ε-transitions in the augmented automaton.

Definition 3.2 (Product MDPs) Given an augmented tLDGBA $\bar{B}_\varphi$ and an MDP M, a tuple $M \otimes \bar{B}_\varphi = M^\otimes = (S^\otimes, A^\otimes, s^\otimes_{init}, P^\otimes, \delta^\otimes,\ \mathcal{F}^\otimes)$ is a product MDP, where

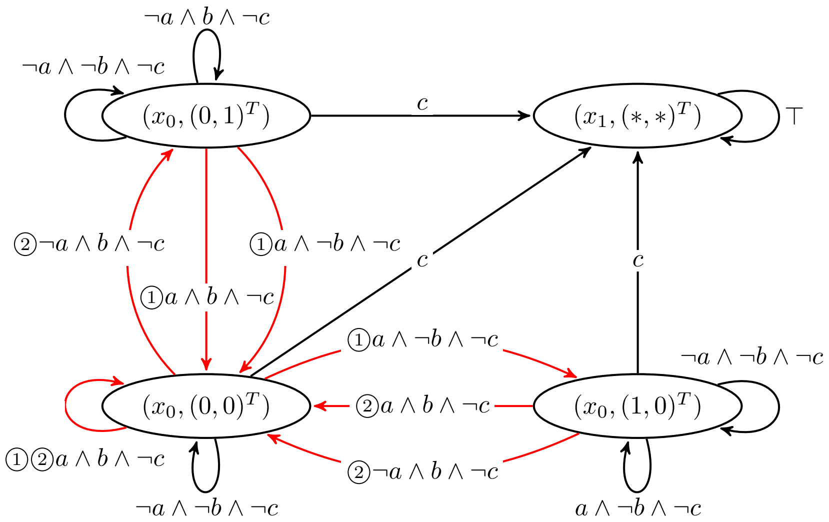

Fig. 3.2 The augmented automaton for the tLDGBA in Fig. 3.1 recognizing the LTL formula $\mathbf{GF}a \wedge \mathbf{GF}b \wedge \mathbf{G}\neg c$, where the initial state is $(x_0, (0,0)^T)$. Red arcs are accepting transitions that are numbered in accordance with the accepting sets they belong to. All states corresponding to x_1 are merged into $(x_1, (*,*)^T)$.

$S^{\otimes} = S \times \bar{X}$ is the finite set of states; $A^{\otimes}$ is the finite set of actions such that $A^{\otimes} = A \cup \{\varepsilon_{\bar{x}'}; \exists \bar{x}' \in X \text{ s.t. } (\bar{x}, \varepsilon, \bar{x}') \in \bar{\delta}\}$, where $\varepsilon_{\bar{x}'}$ is the action for the ε-transition to the state $\bar{x}' \in \bar{X}$; $s^{\otimes}_{init} = (s_{init}, \bar{x}_{init})$ is the initial state; $P^{\otimes} : S^{\otimes} \times S^{\otimes} \times A^{\otimes} \rightarrow [0,1]$ is the transition probability function defined as

$$P^{\otimes}(s^{\otimes\prime}|s^{\otimes}, a) = \begin{cases} P(s'|s,a) & \text{if } (\bar{x}, L((s,a,s')), \bar{x}') \in \bar{\delta}, a \in \mathcal{A}(s) \\ 1 & \text{if } s = s', (\bar{x}, \varepsilon, \bar{x}') \in \bar{\delta}, a = \varepsilon_{\bar{x}'}, \\ 0 & \text{otherwise,} \end{cases}$$

where $s^{\otimes} = (s, (x, v))$ and $s^{\otimes\prime} = (s', (x', v'))$; $\delta^{\otimes} = \{(s^{\otimes}, a, s^{\otimes\prime}) \in S^{\otimes} \times A^{\otimes} \times S^{\otimes}; P^{\otimes}(s^{\otimes\prime}|s^{\otimes}, a) > 0\}$ is the set of transitions; and $\mathcal{F}^{\otimes} = \{\bar{F}^{\otimes}_1, \ldots, \bar{F}^{\otimes}_n\}$ is the acceptance condition, where $\bar{F}^{\otimes}_i = \{((s,\bar{x}), a, (s', \bar{x}')) \in \delta^{\otimes}; (\bar{x}, L(s,a,s'), \bar{x}') \in \bar{F}_i\}$ for each $i \in \{1, \ldots, n\}$.

Definition 3.3 (Reward assignments) The reward function $\mathcal{R} : S^{\otimes} \times A^{\otimes} \times S^{\otimes} \rightarrow \mathbb{R}_{\geq 0}$ is defined as

$$\mathcal{R}(s^{\otimes}, a, s^{\otimes\prime}) = \begin{cases} r_p \text{ if } \exists i \in \{1, \ldots, n\},\ (s^{\otimes}, a, s^{\otimes\prime}) \in \bar{F}^{\otimes}_i, \\ 0 \ \ \text{otherwise,} \end{cases}$$

where r_p is a positive value.

Under the product MDP $M^{\otimes}$ and the reward function $\mathcal{R}$, which is based on the acceptance condition of $M^{\otimes}$, we show that if there exists a positional policy π satisfying the LTL specification φ, maximizing the expected discounted reward produces a policy satisfying φ.

For a Markov chain $MC_{\pi}^{\otimes}$ induced by a product MDP $M^{\otimes}$ with a positional policy π, let $S_{\pi}^{\otimes} = T_{\pi}^{\otimes} \cup R_{\pi}^{\otimes 1} \cup \ldots \cup R_{\pi}^{\otimes h}$ be the set of states in $MC_{\pi}^{\otimes}$, where $T_{\pi}^{\otimes}$ is the set of transient states and $R_{\pi}^{\otimes i}$ is the recurrent class for each $i \in \{1, \ldots, h\}$, and let $R(MC_{\pi}^{\otimes})$ be the set of all recurrent classes in $MC_{\pi}^{\otimes}$. Let $\delta_{\pi,i}^{\otimes}$ be the set of transtions in a recurrent class $R_{\pi}^{\otimes i}$, namely $\delta_{\pi,i}^{\otimes} = \{(s^{\otimes}, a, s^{\otimes\prime}) \in \delta^{\otimes}; s^{\otimes} \in R_{\pi}^{\otimes i},\ P^{\otimes}(s^{\otimes\prime}|s^{\otimes}, a) > 0\}$, and let $P_{\pi}^{\otimes} : S_{\pi}^{\otimes} \times S_{\pi}^{\otimes} \rightarrow [0, 1]$ be the transition probability under π.

Lemma 3.1 For any policy π and any recurrent class $R_{\pi}^{\otimes i}$ in the Markov chain $MC_{\pi}^{\otimes}$, $MC_{\pi}^{\otimes}$ satisfies one of the following conditions.

1. $\delta_{\pi,i}^{\otimes} \cap \bar{F}_j^{\otimes} \neq \emptyset$, $\forall j \in \{1, \ldots, n\}$,
2. $\delta_{\pi,i}^{\otimes} \cap \bar{F}_j^{\otimes} = \emptyset$, $\forall j \in \{1, \ldots, n\}$.

The proof of Lemma 3.1 is shown in Appendix A.

Lemma 3.1 implies that, for an LTL formula φ if a path ρ under a policy π does not satisfy φ, then the agent obtains no reward in recurrent classes; otherwise there exists at least one recurrent class where the agent obtains rewards infinitely often.

Theorem 3.1 Let $M^{\otimes}$ be the product MDP corresponding to an MDP M and an LTL formula φ. If there exists a positional policy satisfying φ, then there exists a discount factor γ^* such that any algorithm that maximizes the expected reward with $\gamma > \gamma^*$ will find a positional policy satisfying φ.

The proof of Theorem 3.1 is shown in Appendix A.

We show the overall procedure of our proposed method in Algorithm 1. We employ Q-learning in Algorithm 1, but any algorithms maximizing the expected discounted reward can be applied to our proposed method.

3.2 Example

In this section, we apply the proposed method to a path planning problem of a robot in an environment consisting of eight rooms and one corridor as shown in Fig. 3.3. The state s_7 is the initial state and the action space is specified with $\mathcal{A}(s) = \{Right, Left, Up, Down\}$ for any state $s \neq s_4$ and $\mathcal{A}(s_4) = \{to\_s_0, to\_s_1, to\_s_2, to\_s_3, to\_s_5, to\_s_6, to\_s_7, to\_s_8\}$, where to_s_i means attempting to go to the state s_i for $i \in \{0,\ 1,\ 2,\ 3,\ 5,\ 6,\ 7,\ 8\}$. The robot moves in the intended direction with probability 0.9 and it stays in the same state with probability 0.1 if it is in the state s_4. In the states other than s_4, it moves in the intended

Algorithm 1 RL-based synthesis of control policy on the MDP with the augmented tLDBA.

Input: LTL formula φ and MDP M

Output: Optimal policy π^* on the product MDP $M^\otimes$

Translate φ into tLDBA B_φ.
Augment B_φ to $\bar{B}_\varphi$.
Construct the product MDP $M^\otimes$ of M and $\bar{B}_\varphi$.
Initialize $Q : S^\otimes \times A^\otimes \to \mathbb{R}_{\geq 0}$.
Initialize episode length T.
while Q is not converged **do**
  $s^\otimes \leftarrow (s_{init}, (x_{init}, \mathbf{0}))$.
  for $t = 1$ to T **do**
    Choose the action a by a policy π.
    Observe the next state $s^{\otimes\prime}$.
    $Q(s^\otimes, a) \leftarrow Q(s^\otimes, a) + \alpha\{\mathcal{R}(s^\otimes, a, s^{\otimes\prime}) + \gamma \max_{a'} Q(s^{\otimes\prime}, a') - Q(s^\otimes, a)\}$
    $s^\otimes \leftarrow s^{\otimes\prime}$
  end for
end while

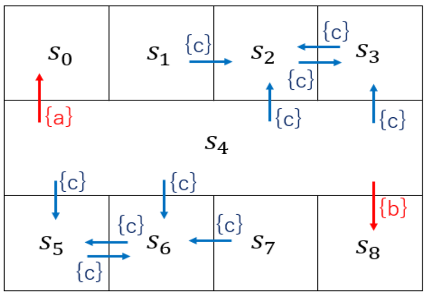

Fig. 3.3 The environment consisting of eight rooms and one corridor. Red arcs are the transitions that we want to occur infinitely often, while blue arcs are the transitions that we never want to occur. s_7 is the initial state.

direction with probability 0.9 and it moves in the opposite direction with probability 0.1. If the robot tries to go to outside the environment, it stays in the same state. The labeling function is as follows.

$$L((s, act, s')) = \begin{cases} \{c\} & \text{if } s' = s_i,\ i \in \{2, 3, 5, 6\}, \\ \{a\} & \text{if } (s, act, s') = (s_4, to\_s_0, s_0), \\ \{b\} & \text{if } (s, act, s') = (s_4, to\_s_8, s_8), \\ \emptyset & \text{otherwise.} \end{cases}$$

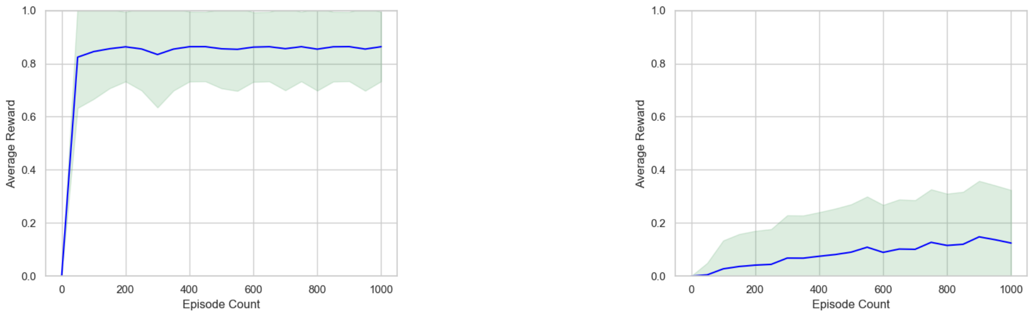

Fig. 3.4 The mean of average reward in each episode for 20 learning sessions obtained from our proposed method (left) and the method using tLDBA (right). They are plotted per 50 episodes and the green areas represent the range of standard deviations.

In the example, the robot tries to take two transitions that we want to occur infinitely often, represented by arcs labeled by $\{a\}$ and $\{b\}$, while avoiding unsafe transitions represented by the arcs labeled by $\{c\}$. This is formally specified by the LTL formula given by (3.1). The LTL formula requires the robot to keep on entering the two rooms s_0 and s_8 from the corridor s_4 regardless of the order of entries, while avoiding entering the four rooms s_2, s_3, s_5, and s_6. The tLDGBA $B_\varphi = (X, x_{init}, \Sigma, \delta, \mathcal{F})$ and its augmented automaton $\bar{B}_\varphi = (\bar{X}, \bar{x}_{init}, \bar{\Sigma}, \bar{\delta}, \bar{\mathcal{F}})$ are shown in Figs. 3.1 and 3.2, respectively.

Through the above scenario, we compare our approach with 1) a case where we first convert the tLDGBA into a tLDBA, for which the augmentation makes no change, and thus a reward function in Definition 3.3 is based on a single accepting set; and 2) the method using a reward function based on the accepting frontier function [14, 16]. For the three methods, we use Q-learning[*1] with ε-greedy policy and gradually reduce ε to 0 to learn an optimal policy asymptotically. We set the positive reward $r_p = 2$, the epsilon greedy parameter $\varepsilon = \frac{0.95}{n_t(s^\otimes)}$, where $n_t(s^\otimes)$ is the number of visits to state $s^\otimes$ within t time steps [21], and the discount factor $\gamma = 0.95$. The learning rate α varies in accordance with *the Robbins-Monro condition*. We train the agent in 10000 iterations and 1000 episodes for 20 learning sessions.

Results

1) Fig. 3.4 shows the average rewards obtained by our proposed method and the case using a tLDBA B'_φ converted from φ, respectively. Both methods eventually acquire an optimal policy satisfying φ. As shown in Fig. 3.4, however, our proposed method converges faster. This is because the order of entrance to the rooms s_0 and s_8 is determined according to

[*1] We employ Q-learning here but any algorithm that maximizes the discounted expected reward can be applied to our proposed method.

Fig. 3.5 The optimal policy obtained from our proposed method (left) and the method in [14, 16] (right).

the tLDBA. Moreover, the number of transitions with a positive reward in $\bar{B}_\varphi$ is larger than that in B'_φ.

2) We use the accepting frontier function [14, 16] for the tLDGBA $Acc : \delta \times 2^\delta \to 2^\delta$. Initializing a set of transitions $\mathbb{F}$ with the set of the all accepting transitions in B_φ, the function receives the transition (x, σ, x') that occurs and the set $\mathbb{F}$. If (x, σ, x') is in $\mathbb{F}$, then Acc removes the accepting sets containing (x, σ, x') from $\mathbb{F}$. For the product MDP of the MDP M and the tLDGBA B_φ, the reward function is based on the removed sets of B_φ.

Fig. 3.5 shows the optimal policies obtained by our proposed method and the method in [14, 16]*2, respectively. The policy obtained by the method with the accepting frontier function fails to satisfy the LTL specification because it is impossible with B_φ shown in Fig. 3.1 that the transitions labeled with $\{a\}$ and $\{b\}$ occur from s_4 infinitely often by any positional policy. More specifically, the state of B_φ is always x_0 while the agent does not move to bad states s_2, s_3, s_5, and s_6. Whenever the agent is in s_4, therefore, the product MDP is always in (s_4, x_0) unless one of the bad states are visited. Thus, the agent cannot visit both of s_0 and s_8 by a deterministic action selection at s_4. On the other hand, our proposed method can recognize the previous visits. Thus, our proposed method can synthesize a positional policy satisfying φ on the product MDP, while the method in [14, 16] cannot. In order to obtain a positional policy satisfying the LTL formula φ with the method in [14, 16] in this example, we have to refine the tLDGBA shown in Fig. 3.1 heuristically, e.g., by adding states with which the order of occurrences of accepting transitions is recognized.

*2 We obtain the same result even with a state-based LDGBA.

Chapter 4

Reinforcement learning based supervisor synthesis for LTL specifications

4.1 Stochastic Discrete Event Systems

We represent a stochastic discrete event system (DES) as an MDP.

Definition 4.1 We represent a probabilistic discrete event system (DES) as a labeled Markov decision process (MDP). A DES is a tuple $D = (S, E, P_T, P_E, s_{init}, AP, L)$, where S is a finite set of states; E is a finite set of events; $P_T : S \times S \times E \to [0, 1]$ is a transition probability; $P_E : E \times S \times 2^E \to [0, 1]$ is the probability of an event occurrence under a state $s \in S$ and a subset $\pi \in \mathcal{E}(s)$; for any $(s', s, \pi) \in S \times S \times 2^E$, we define the probability $P : S \times S \times 2^E \to [0, 1]$ such that $P(s'|s, \pi) = \sum_{e \in \pi} P_E(e|s, \pi) P_T(s'|s, e)$; $s_{init} \in S$ is the initial state; AP is a finite set of atomic propositions; and $L : S \times E \times S \to 2^{AP}$ is a labeling function that assigns a set of atomic propositions to each transition $(s, e, s') \in S \times E \times S$. Let $\mathcal{E}(s) = \{e \in E; \exists s' \in S \text{ s.t. } P_T(s'|s, e) \neq 0\}$. Note that $\sum_{s' \in S} P_T(s'|s, e) = 1$ holds for any state $s \in S$ and event $e \in E$, $\Sigma_{e \in \pi} P_E(e|s, \pi) = 1$ holds for any state $s \in S$ and a subset $\pi \in 2^E$, and $\Sigma_{s' \in S} P(s'|s, \pi) = 1$ holds for any state $s \in S$ and a subset $\pi \in 2^E$. We assume that E is partitioned into the set of controllable events E_c and the set of uncontrollable events E_{uc} such that $E_c \cup E_{uc} = E$ and $E_c \cap E_{uc} = \emptyset$. Note that each event e occurs probabilistically depending on only the current state and the subset of feasible events at the state given by a controller.

In the DES D, an infinite path starting from a state $s_0 \in S$ is defined as a sequence $\rho = s_0 \pi_0 e_0 s_1 \ldots \in S(2^E E S)^\omega$ such that $P_T(e_i|s_i, \pi_i) > 0$ and $P_T(s_{i+1}|s_i, e_i) > 0$ for any $i \in \mathbb{N}_0$. A finite path is a finite sequence in $S(2^E E S)^*$. In addition, we sometimes represent ρ as ρ_{init} to emphasize that ρ starts from $s_0 = s_{init}$. For a path $\rho = s_0 \pi_0 e_0 s_1 \ldots$, we define the corresponding labeled path $L(\rho) = L(s_0, e_0, s_1)L(s_1, e_1, s_2) \ldots \in (2^{AP})^{\omega}$. $InfPath^D$ (resp., $FinPath^D$) is defined as the set of infinite (resp., finite) paths starting from $s_0 = s_{init}$ in the DES D. For each finite path ρ, $last(\rho)$ denotes its last state.

We define the supervisor as a controller for the DES that restricts the behaviors of the DES to satisfy a given specification.

Definition 4.2 (Supervisor) For a DES D, a supervisor $SV : FinPath^D \to 2^E$ is defined as a mapping that maps each finite path to a set of enabled events at the finite path and we call the set a control pattern. A supervisor is positional if $SV(\rho) = SV(last(\rho))$ for any $\rho \in InfPath^D$. Note that the relation $E_{uc} \subset SV(\rho) \subset E$ holds for any $\rho \in FinPath^D$. Let $InfPath^D_{SV}$ (resp., $FinPath^D_{SV}$) be the set of infinite (resp., finite) paths starting from $s_0 = s_{init}$ in the DES D under a supervisor SV. The behavior of the DES D under the supervisor SV is defined on a probability space $(InfPath^D_{SV}, \mathcal{F}_{InfPath^D_{SV}}, Pr^D_{SV})$.

For any supervisor SV, the probability of all paths starting from s_{init} on the MDP M that satisfy an LTL formula φ under the supervisor SV, or the satisfaction probability under the supervisor is defined as follows.

$$Pr^D_{SV}(s_{init} \models \varphi) := Pr^D_{SV}(\{\rho_{init} \in InfPath^D_{SV}; \rho_{init} \models \varphi\}).$$

We say that an LTL formula φ is satisfied by a positional supervisor SV if

$$Pr^D_{SV}(s_{init} \models \varphi) > 0.$$

We consider the objective function similar to the *Bellman optimality function* defined by Definition 2.5.

Definition 4.3 (Optimal value function for DESs) From the view point of reinforcement learning, the DES can be interpreted as the environment controlled by the supervisor and the supervisor can be interpreted as the policy. We introduce the two following assumptions.

1. The relative frequency of occurrence of each event does not depend on the control pattern.
2. We define a reward function $\mathcal{R} : S \times 2^E \times E \times S \to \mathbb{R}$ and the reward $\mathcal{R}$ can be decomposed into $\mathcal{R}_1$ and $\mathcal{R}_2$. The first reward $\mathcal{R}_1 : S \times 2^E \to \mathbb{R}$ is determined by the control pattern selected by the supervisor, which depends on only the control pattern and the current state. The second reward $\mathcal{R}_2 : S \times E \times S \to \mathbb{R}$ is determined by the occurrence of an event and the corresponding state transition. For any $(s, \pi, e, s') \in S \times 2^E \times E \times S$, we then have

$$\mathcal{R}(s, \pi, e, s') = \mathcal{R}_1(s, \pi) + \mathcal{R}_2(s, e, s'). \tag{4.1}$$

Under the above assumptions, we have the following *Bellman optimality equation.*

$$
\begin{aligned}
Q^*(s,\pi) &= \sum_{s'\in S} P(s'|s,\pi)\left\{\mathcal{R}(s,\pi,e,s') + \gamma \max_{\pi'\in 2^{\mathcal{E}(s')}} Q^*(s',\pi')\right\} \\
&= \sum_{s'\in S}\sum_{e\in\pi} P_E(e|s,\pi)P_T(s'|s,e)\left\{\mathcal{R}_1(s,\pi) + \mathcal{R}_2(s,e,s') + \gamma \max_{\pi'\in 2^{\mathcal{E}(s')}} Q^*(s',\pi')\right\} \\
&= \mathcal{R}_1(s,\pi) + \sum_{e\in\pi} P_E(e|s,\pi)\sum_{s'\in S} P_T(s'|s,e)\left\{\mathcal{R}_2(s'|s,e) + \gamma \max_{\pi'\in 2^{\mathcal{E}(s')}} Q^*(s',\pi')\right\},
\end{aligned} \tag{4.2}
$$

where $\gamma \in [0,1)$.

We introduce the following function. $T^* : S \times E \to \mathbb{R}$ such that

$$
T^*(s,e) = \sum_{s'\in S} P_T(s'|s,e)\left\{\mathcal{R}_2(s'|s,e) + \gamma \max_{\pi'\in 2^{\mathcal{E}(s')}} Q^*(s',\pi')\right\}. \tag{4.3}
$$

We then have

$$
Q^*(s,\pi) = \mathcal{R}_1(s,\pi) + \sum_{e\in\pi} P_E(e|s,\pi)T^*(s,e). \tag{4.4}
$$

Definition 4.4（Optimal supervisor） We define an optimal supervisor SV^* as follows. For any state $s \in S$,

$$
SV^*(s) = \pi \in \arg\max_{\pi\in\mathcal{E}(s)} Q^*(s,\pi), \tag{4.5}
$$

4.2 Product DESs

Definition 4.5 Given an augmented tLDBA $\bar{B}_\varphi = (\bar{X}, \bar{x}_{init}, \bar{\Sigma}, \bar{\delta}, \bar{\mathcal{F}})$ and a DES D, a tuple $D \otimes \bar{B}_\varphi = D^\otimes = (S^\otimes, E^\otimes, s^\otimes_{init}, P^\otimes_T, P^\otimes_E, \delta^\otimes, \mathcal{F}^\otimes)$ is a product DES, where $S^\otimes = S \times \bar{X}$ is the finite set of states and we represent s and $\bar{x}$ corresponding with $s^\otimes = (s, \bar{x}) \in S^\otimes$ as $[\![s^\otimes]\!]_s$ and $[\![s^\otimes]\!]_q$, respectively; $E^\otimes = E \cup \{\varepsilon_{\bar{x}'}; \exists \bar{x}' \text{s.t.} (\bar{x}, \varepsilon, \bar{x}') \in \bar{\delta}\}$ is the finite set of events, where $\varepsilon_{\bar{x}'}$ is the event that represents an ε-transition to $\bar{x}' \in \bar{X}$; $s^\otimes_{init} = (s_{init}, \bar{x}_{init})$ is the initial states, $P^\otimes_T : S^\otimes \times S^\otimes \times E^\otimes \to [0,1]$ is the transition probability defined as

$$
P^\otimes_T(s^{\otimes\prime}|s^\otimes, e) = \begin{cases} P_T(s'|s,e) & \text{if } (\bar{x}, L((s,e,s')), \bar{x}') \in \bar{\delta}, e \in \mathcal{E}(s) \\ 1 & \text{if } s = s', (\bar{x}, \varepsilon, \bar{x}') \in \delta, e = \varepsilon_{\bar{x}'} \\ 0 & \text{otherwise,} \end{cases}
$$

where $s^\otimes = (s,(x,v))$ and $s^{\otimes\prime} = (s',(x',v'))$. $P^\otimes_E : E^\otimes \times S^\otimes \times 2^{E^\otimes} \to [0,1]$ is the probability of the occurrence of the event defined as $P^\otimes_E(e|s^\otimes,\pi) = P_E(e|s,\pi)$, $\delta^\otimes = \{(s^\otimes, e, s^{\otimes\prime}) \in$

$S^{\otimes} \times E^{\otimes} \times S^{\otimes}; P_T^{\otimes}(s^{\otimes\prime}|s^{\otimes}, e) > 0\}$ is the set of transitions, and $\mathcal{F}^{\otimes} = \{\bar{F}_1^{\otimes}, \ldots, \bar{F}_n^{\otimes}\}$ is the acceptance condition, where $\bar{F}_i^{\otimes} = \{((s, \bar{x}), e, (s', \bar{x}')) \in \delta^{\otimes} \ ; \ (\bar{x}, L(s, e, s'), \bar{x}') \in \bar{F}_i\}$ for each $i \in \{1, \ldots, n\}$.

Definition 4.6 The two reward functions $\mathcal{R}_1 : S^{\otimes} \times 2^{E^{\otimes}} \to \mathbb{R}$ and $\mathcal{R}_2 : S^{\otimes} \times E^{\otimes} \times S^{\otimes} \to \mathbb{R}$ are defined as follows.

$$\mathcal{R}_1(s^{\otimes}, \pi) = \begin{cases} r_n|\pi| & \text{if } [\![s^{\otimes}]\!]_q \notin \mathit{SinkSet}, \\ 0 & \text{otherwise}, \end{cases} \tag{4.6}$$

where $|E|$ means the number of elements in the set E and r_n is a positive value.

$$\mathcal{R}_2(s^{\otimes}, e, s^{\otimes\prime}) = \begin{cases} r_p & \text{if } \exists j \in \{1, \ldots, n\},\ (s^{\otimes}, e, s^{\otimes\prime}) \in \bar{F}_j^{\otimes}, \\ r_{sink} & \text{if } [\![s^{\otimes\prime}]\!]_q \in \mathit{SinkSet}, \\ 0 & \text{otherwise}, \end{cases} \tag{4.7}$$

where r_p and r_{sink} are the positive and negative value, respectively.

4.3 Learning Algorithm

We make the supervisor learn how to give the control patterns to satisfy an LTL specification while keeping costs associated with disabled events low. We use Q-learning to estimate the function T^*. We then use Bayesian inference to robustly estimate the probability P_E. For the inference, we model P_E as a categorical distribution as $p^k_{s,\pi,e}$, where $p^k_{s,\pi,e}$ represents the estimated probability of $P_E(e|s, \pi)$ at the time step k and the prior distribution $\phi^k_{s,\pi}$ for the distribution of the parameter of $p^k_{s,\pi,e}$ is defined as a Dirichlet distribution.

In the following, we distinguish events by numbering them as $\{e^1, \ldots, e^{|E|}\}$. In order to reflect the events disabled by the supervisor on the estimated probability of an event occurrence, we introduce the function $\mathit{RestProb} : (0,1)^{|E|} \times 2^E \to [0,1]^{|E|}$ defined by

$$\mathit{RestProb}(\phi^k_{s,\pi}, \pi)_i = \begin{cases} \dfrac{\phi^{k,i}_{s,\pi}}{\sum_{e^j \in \pi} \phi^j_{s,\pi}} & \text{if } e^i \in \pi, \\ 0 & \text{otherwise}, \end{cases} \tag{4.8}$$

where $\phi^{k,i}_{s,\pi}$ is the i-th element of $\phi^k_{s,\pi}$ and $\mathit{RestProb}(\phi^k_{s,\pi}, \pi)_i$ is the i-th element of $\mathit{RestProb}(\phi^k_{s,\pi}, \pi)$.

We denote the probability vector of an event occurrence at the time step k as $p^k_{s,\pi} = (p^k_{s,\pi,e^1}, \ldots, p^k_{s,\pi,e^{|E|}})$, where $s \in S$ and $\pi \in 2^{\mathcal{E}(s)}$ is the state and the control pattern at the time step k. Let $n^k_{s,\pi,e}$ be the number of the occurrence of the event $e \in E$ up to the time step k at the state $s \in S$ under the control pattern $\pi \in 2^{\mathcal{E}(s)}$ and let $n^k_{s,\pi} = (n^k_{s,\pi,e_1}, \ldots, n^k_{s,\pi,e_{|E|}})$. We sample the parameter vector $\phi^k_{s,\pi}$ of the posterior distribution of an event occurrence from the Dirichlet distribution $Dir(\cdot|n^k_{s,\pi})$. Then, we obtain the estimated probability vector $p^k_{s,\pi}$ of an event occurrence by $RestProb$ from the sampled parameter $\phi^k_{s,\pi}$ and the control pattern π.

The overall procedure of the inference is shown in Algorithm 2.

Algorithm 2 P_E inference.

Input: the vector of the event occurrence count $n^k_{s,\pi}$ up to the time step k at the state $s \in S$ with the control pattern $\pi \in 2^{\mathcal{E}(s)}$.

Output: the posterior distribution $p^k_{s,\pi}$

1: $\phi^k_{s,\pi} \sim Dir(\cdot|n^k_{s,\pi})$

2: $p^k_{s,\pi} = RestProb(\phi^k_{s,\pi}, \pi)$

Under the estimation of P_E, we use TD-learning to estimate Q^* with the TD-error defined as $\mathcal{R}_1(s^{\otimes}, \pi) + \sum_{e \in \pi} p_{[\![s^{\otimes}]\!]_s,\pi,e} T(s^{\otimes}, e) - Q(s^{\otimes}, \pi)$.

We show the overall procedure of the learning algorithm in Algorithm 3.

Algorithm 3 RL-based synthesis of a supervisor satisfying the given LTL specification.

Input: LTL formula φ, DES M

Output: optimal supervisor SV^* on the product DES $M^{\otimes}$

Convert φ into tLDGBA B_{φ}.
Augment B_{φ} to $\bar{B}_{\varphi}$.
Construct the product DES $M^{\otimes}$ of M and $\bar{B}_{\varphi}$.
Initialize $T : S^{\otimes} \times E^{\otimes} \to \mathbb{R}$.
Initialize $Q : S^{\otimes} \times 2^{E^{\otimes}} \to \mathbb{R}$.
Initialize $n : S \times 2^{E} \times E \to \mathbb{R}$.
initialize $\xi : S \times 2^{E} \to \mathbb{R}$.
Initialize episode length L.
while Q is not converged **do**
    $s^{\otimes} \leftarrow (s_{init}, (x_{init}, \mathbf{0}))$.
    $t \leftarrow 0$
    while $t < L$ and $[\![s^{\otimes}]\!]_q \notin SinkSet$ **do**
        Choose the control pattern $\pi \in 2^{\mathcal{E}(s^{\otimes})}$ by the supervisor SV.
        Observe the occurrence of the event $e \in E$.
        Observe the next state $s^{\otimes\prime}$.
        $T(s^{\otimes}, e) \leftarrow (1-\alpha)T(s^{\otimes}, e) + \alpha\{\mathcal{R}_2(s^{\otimes}, e, s^{\otimes\prime}) + \gamma \max_{\pi' \in 2^{\mathcal{E}(s^{\otimes\prime})}} Q(s^{\otimes\prime}, \pi')\}$
        $n([\![s^{\otimes}]\!]_s, \pi, e) \leftarrow n([\![s^{\otimes}]\!]_s, \pi, e) + 1$
        Obtain $p_{[\![s^{\otimes}]\!]_s, \pi}$ from n by the P_E inference.
        $Q(s^{\otimes}, \pi) = (1-\beta)Q(s^{\otimes}, \pi) + \beta\{\mathcal{R}_1(s^{\otimes}, \pi) + \sum_{e \in \pi} p_{[\![s^{\otimes}]\!]_s, \pi, e} T(s^{\otimes}, e)\}$
        $s^{\otimes} \leftarrow s^{\otimes\prime}$
        $t \leftarrow t + 1$
    end while
end while

4.4 Example

We evaluate the algorithm by the maze of the cat and the mouse shown in Fig. 4.1. At the beginning, we define the settings for the example. The corresponding DES is as follows. The state set is $S = \{(s^{cat}, s^{mouse}); s^{cat}, s^{mouse} \in \{s_0, s_1, s_2, s_3\}\}$. The set of events (to open the corresponding door) is $E = \{m_0, m_1, m_2, m_3, c_0, c_1, c_2, c_3\}$, where $E_c = \{m_0, m_1, m_2, m_3, c_0, c_1, c_2\}$ and $E_{uc} = \{c_3\}$ and $\mathcal{E}(s) = E$ for any $s \in S$. The initial state is $s_{init} = (s_0, s_2)$. If the door of the room with the cat (resp., mouse) opens, the cat (resp., mouse) moves, with probability 0.95, to the room next to the room through the door or stays in the same room with probability 0.05. Otherwise, the cat (resp., mouse) stays in the same room with probability 1. The labeling function is

$$L((s,a,s')) = \begin{cases} \{a\} & \text{if } s'_c = s_1, \\ \{b\} & \text{if } s'_m = s_1, \\ \{c\} & \text{if } s'_c = s'_m, \\ \emptyset & \text{otherwise,} \end{cases}$$

where s'_c and s'_m is the next room where the cat and the mouse is, respectively, i.e., $s' = (s'_c, s'_m)$.

In the example, we want the supervisor to learn to give control patterns satisfying that the cat and the mouse take the food in the room 1 (s_1) avoiding they are in same room simultaneously. This is formally specified by the following LTL formula given by Eq. 3.1. The augmented tLDGBA $\bar{B}_\varphi = (\bar{X}, \bar{x}_{init}, \bar{\Sigma}, \bar{\delta}, \bar{\mathcal{F}})$ corresponding to φ is shown in Fig. 3.2. $\bar{B}_\varphi$ has the acceptance condition of two accepting sets.

We use an ε-greedy policy and gradually reduce ε to 0 to learn an optimal supervisor asymptotically. We set the rewards $r_p = 10$, $r_n = 0.1, 0.7$, and 1.2, and $r_{sink} = -1000$; the epsilon greedy parameter $\varepsilon = \frac{1}{\sqrt{episode}}$, where *episode* is the number of the current episode; and the discount factor $\gamma = 0.99$. The learning rates α and β vary in accordance with *the Robbins-Monro condition*. We train supervisors 5000 iterations and 15000 episodes.

Results

Fig. 4.2 shows the estimated optimal state values of the initial state $V(s^{\otimes}_{init})$ with $r_n = 0.1, 0.7$, and 1.2, respectively, for each episode when learning 5000 iterations and 15000 episodes by the algorithm 3. Fig. 4.3 shows the average rewards from $\mathcal{R}_2$ and the average rewards from $\mathcal{R}_1$ with $r_n = 0.1, 0.7$, and 1.2, respectively, of 5000 iterations and 1000 episodes by the supervisor obtained from the learning.

Fig. 4.2 shows the three supervisors becomes optimal as the episode progresses. Fig. 4.3 suggests that the three supervisors obtained from the learning satisfy φ and the safety constraint is satisfied with probability 1 under the supervisors. The latter is implied by the stable average rewards for the LTL formula φ. Furthermore, Fig. 4.3 shows that there is a trade-off between the frequency of visits to accepting sets of the product MDP of the given MDP and the augmented tLDGBA converted from φ and the number of enabling events.

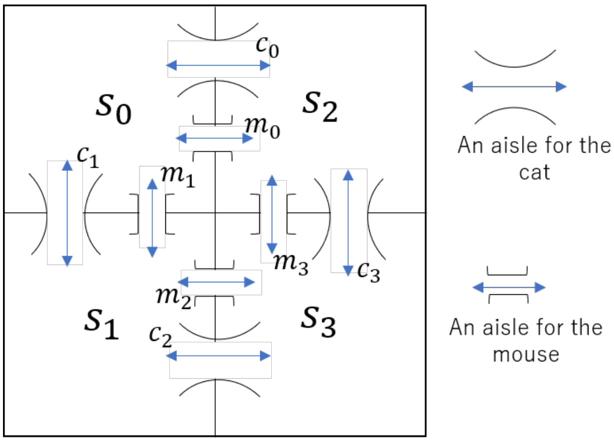

Fig. 4.1 The maze of the cat and the mouse. the initial state of the cat and the mouse is s_0 and s_2, respectively. the food for them is in the room 1 (s_1).

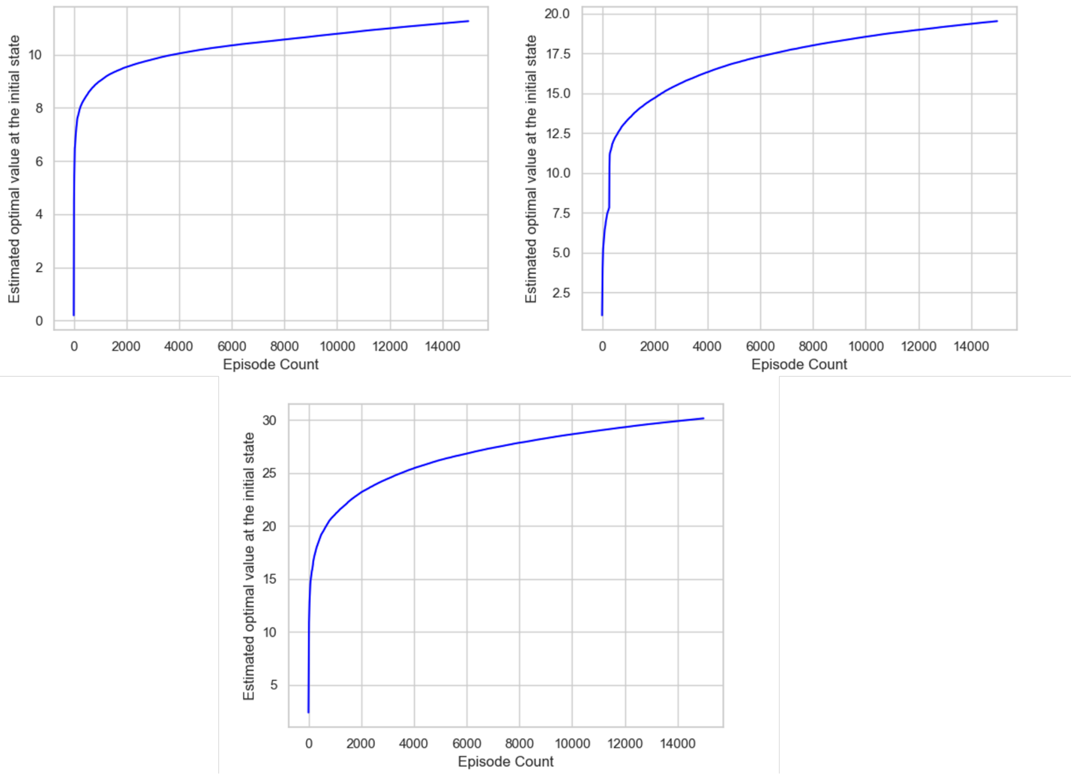

Fig. 4.2 The estimated optimal state values at the initial state $V(s^{\otimes}_{init})$ with $r_n = 0.1$ (left above), $r_n = 0.7$ (right above), and $r_n = 1.2$ (below) when using Algorithm 3.

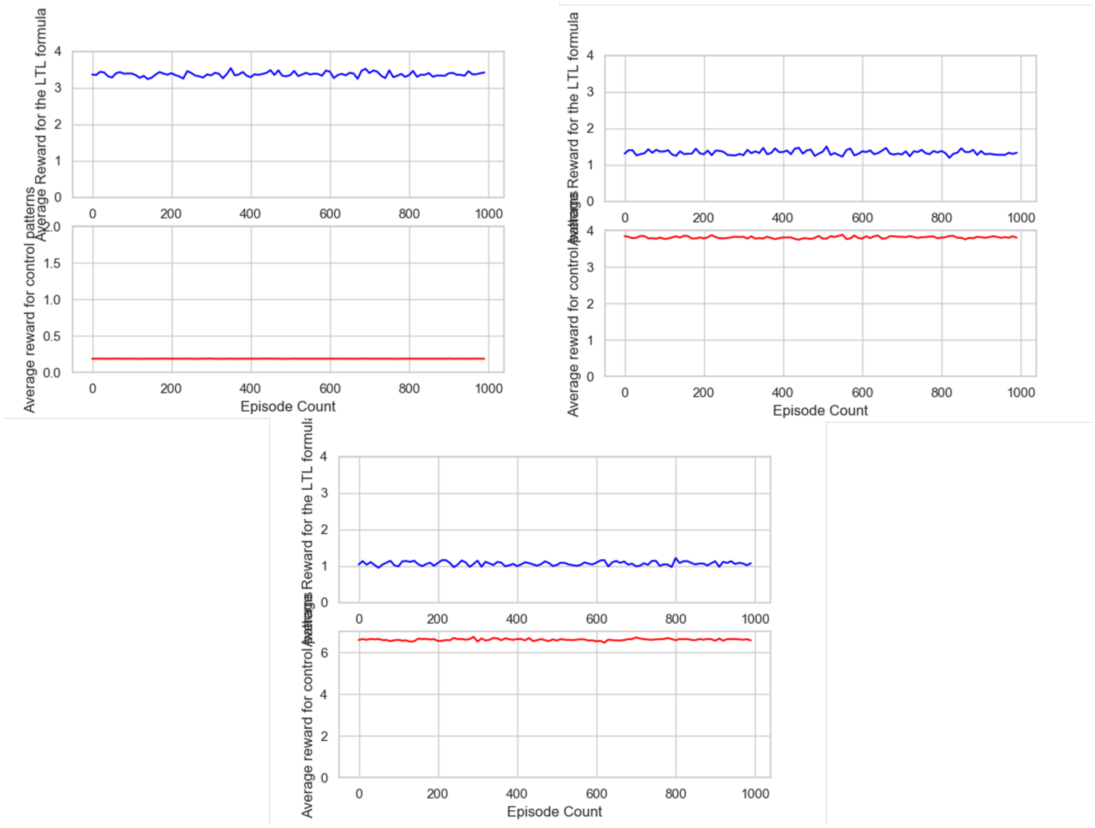

Fig. 4.3 The average rewards for $\mathcal{R}_1$ and average rewards for $\mathcal{R}_2$ by the supervisor obtained from the learning with $r_n = 0.1$ (left above), $r_n = 0.7$ (right above), and $r_n = 1.2$ (below).

Chapter 5

Conclusions

In this thesis, we proposed a novel RL-based method for the synthesis of a controller for an LTL specification using a limit-deterministic generalized Büchi automaton. The proposed method improved the learning performance compared to existing methods and we showed that the proposed method synthesizes a control policy satisfying a given LTL formula with nonzero probability. Then, we proposed a novel RL-based method with an augmented tLDGBA that synthesizes a supervisor satisfying (i) a given LTL formula with nonzero probability and (ii) safety constraints with probability 1. We evaluated the proposed method with an example of the maze of a cat and a mouse. The following arguments are future works. (i) We investigate the method synthesizing a controller that maximizes the satisfaction probability. (ii) We expand our proposed method to a hierarchical control. (iii) We apply our proposed method to multiagent systems.

A

Proofs

Proof （Proof of Proposition3.1） First we show $\mathcal{L}(B) \subseteq \mathcal{L}(\bar{B})$, then show $\mathcal{L}(B) \supseteq \mathcal{L}(\bar{B})$, concluding $\mathcal{L}(B) = \mathcal{L}(\bar{B})$.

- $\mathcal{L}(B) \subseteq \mathcal{L}(\bar{B})$: Consider any $w = \sigma_0\sigma_1 \ldots \in \mathcal{L}(B)$. Then, there exists a run $r = x_0\sigma_0x_1\sigma_1x_2 \ldots \in X(\Sigma X)^\omega$ of B such that $x_0 = x_{init}$ and $inf(r) \cap F_j \neq \emptyset$ for each $F_j \in \mathcal{F}$. Recall that $\Sigma = \bar{\Sigma}$. For the run r, we construct a sequence $\bar{r} = \bar{x}_0\bar{\sigma}_0\bar{x}_1\bar{\sigma}_1\bar{x}_2 \ldots \in \bar{X}(\bar{\Sigma}\bar{X})^\omega$ satisfying $\bar{x}_i = (x_i, v_i)$ and $\bar{\sigma}_i = \sigma_i$ for any $i \in \mathbb{N}$, where
$$v_0 = \mathbf{0} \text{ and } \forall i \in \mathbb{N},\ v_{i+1} = reset\Big(Max\big(v_i, visitf((x_i, \bar{\sigma}_i, x_{i+1}))\big)\Big).$$
Clearly from the construction, we have $(\bar{x}_i, \bar{\sigma}_i, \bar{x}_{i+1}) \in \bar{\delta}$ for any $i \in \mathbb{N}$. Thus, $\bar{r}$ is a run of $\bar{B}$ starting from $\bar{x}_0 = (x_{init}, \mathbf{0}) = \bar{x}_{init}$.
We now show that $inf(\bar{r}) \cap \bar{F}_j \neq \emptyset$ for each $\bar{F}_j \in \bar{\mathcal{F}}$. Since $inf(r) \cap F_j \neq \emptyset$ for each $F_j \in \mathcal{F}$, we have
$$\forall j \in \{1, \ldots, n\},\ inf(\bar{r}) \cap \{\big((x, v), \bar{\sigma}, (x', v')\big) \in \bar{\delta} : visitf((x, \bar{\sigma}, x'))_j = 1\} \neq \emptyset.$$
By the construction of $\bar{r}$, therefore, there are infinitely many indices $l \in \mathbb{N}$ with $v_l = \mathbf{0}$. Let $l_1, l_2 \in \mathbb{N}$ be arbitrary nonnegative integers such that $l_1 < l_2$, $v_{l_1} = v_{l_2} = \mathbf{0}$, and $v_{l'} \neq \mathbf{0}$ for any $l' \in \{l_1 + 1, \ldots, l_2 - 1\}$. Then,
$$\forall j \in \{1, \ldots, n\},\ \exists k \in \{l_1, l_1 + 1, \ldots, l_2 - 1\},\ (x_k, \sigma_k, x_{k+1}) \in F_j \wedge (v_k)_j = 0,$$
where $(v_k)_j$ is the j-th element of v_k. Hence, we have $inf(\bar{r}) \cap \bar{F}_j \neq \emptyset$ for each $\bar{F}_j \in \bar{\mathcal{F}}$, which implies $w \in \mathcal{L}(\bar{B})$.
- $\mathcal{L}(B) \supseteq \mathcal{L}(\bar{B})$: Consider any $\bar{w} \in \bar{\sigma}_0\bar{\sigma}_1 \ldots \in \mathcal{L}(\bar{B})$. Then, there exists a run $\bar{r} = \bar{x}_0\bar{\sigma}_0\bar{x}_1\bar{\sigma}_1\bar{x}_2 \ldots \in \bar{X}(\bar{\Sigma}\bar{X})^\omega$ of $\bar{B}$ such that $\bar{x}_0 = \bar{x}_{init}$ and $inf(\bar{r}) \cap \bar{F}_j \neq \emptyset$ for each $\bar{F}_j \in \bar{\mathcal{F}}$, i.e.,
$$\forall j \in \{1, \ldots, n\},\ \forall k \in \mathbb{N},\ \exists l \geq k,\ ([\![\bar{x}_l]\!]_X, \bar{\sigma}_l, [\![\bar{x}_{l+1}]\!]_X) \in F_j \wedge (\bar{v}_l)_j = 0, \quad \text{(A.1)}$$

where $[\![(x,v)]\!]_X = x$ for each $(x,v) \in \bar{X}$. For the run $\bar{r}$, we construct a sequence $r = x_0\sigma_0 x_1\sigma_1 x_2 \ldots \in X(\Sigma X)^\omega$ such that $x_i = [\![\bar{x}_i]\!]_X$ and $\sigma_i = \bar{\sigma}_i$ for any $i \in \mathbb{N}$. It is clear that r is a run of B starting from $x_0 = x_{init}$. It holds by Eq. (A.1) that $inf(r) \cap F_j \neq \emptyset$ for each $F_j \in \mathcal{F}$, which implies $\bar{w} \in \mathcal{L}(B)$.

$\square$

Proof (Proof of Lemma 3.1) Suppose that $MC_\pi^\otimes$ satisfies neither conditions 1 nor 2. Then, there exist a policy π, $i \in \{1,\ldots,h\}$, and $j_1,\ j_2 \in \{1,\ldots,n\}$ such that $\delta_\pi^{\otimes i} \cap \bar{F}_{j_1}^\otimes = \emptyset$ and $\delta_\pi^{\otimes i} \cap \bar{F}_{j_2}^\otimes \neq \emptyset$. In other words, there exists a nonempty and proper subset $J \in 2^{\{1,\ldots,n\}} \setminus \{\{1,\ldots,n\},\emptyset\}$ such that $\delta_\pi^{\otimes i} \cap \bar{F}_j^\otimes \neq \emptyset$ for any $j \in J$. For any transition $(s,a,s') \in \delta_\pi^{\otimes i} \cap \bar{F}_j^\otimes$, the following equation holds by the properties of the recurrent states in $MC_\pi^\otimes$[18].

$$\sum_{k=0}^{\infty} p^k((s,a,s'),(s,a,s')) = \infty, \tag{A.2}$$

where $p^k((s,a,s'),(s,a,s'))$ is the probability that the transition (s,a,s') reoccurs after it occurs in k time steps. Eq. (A.2) means that all transition in $R_\pi^{\otimes i}$ occurs infinitely often. However, the memory state v is never reset in $R_\pi^{\otimes i}$ by the assumption. This directly contradicts Eq. (A.2). $\square$

Proof (Proof of Theorem 3.1) Suppose that π^* is an optimal policy but does not satisfy the LTL formula φ. Then, for any recurrent class $R_{\pi^*}^{\otimes i}$ in the Markov chain $MC_{\pi^*}^\otimes$ and any accepting set $\bar{F}_j^\otimes$ of the product MDP $M^\otimes$, $\delta_{\pi^*}^{\otimes i} \cap \bar{F}_j^\otimes = \emptyset$ holds by Lemma **??**. Thus, the agent under the policy π^* can obtain rewards only in the set of transient states. We consider the best scenario in the assumption. Let $p^k(s,s')$ be the probability of going to a state s' in k time steps after leaving the state s, and let $Post(T_\pi^\otimes)$ be the set of states in recurrent classes that can be transitioned from states in $T_\pi^\otimes$ by one action. For the initial state s_{init} in the set of transient states, it holds that

$$\begin{aligned} V^{\pi^*}(s_{init}) &= \sum_{k=0}^{\infty} \sum_{s \in T_{\pi^*}^\otimes} \gamma^k p^k(s_{init}, s) \sum_{s' \in T_{\pi^*}^\otimes \cup Post(T_{\pi^*}^\otimes)} P_{\pi^*}^\otimes(s'|s)\mathcal{R}(s,a,s') \\ &\leq r_p \sum_{k=0}^{\infty} \sum_{s \in T_{\pi^*}^\otimes} \gamma^k p^k(s_{init}, s), \end{aligned}$$

where the action a is selected by π^*. By the property of the transient states, for any state $s^\otimes$ in $T_{\pi^*}^\otimes$, there exists a bounded positive value m such that $\sum_{k=0}^\infty \gamma^k p^k(s_{init},s) \leq \sum_{k=0}^\infty p^k(s_{init},s) < m$ [18]. Therefore, there exists a bounded positive value $\bar{m}$ such that $V^{\pi^*}(s_{init}) < \bar{m}$. Let $\bar{\pi}$ be a positional policy satisfying φ. We consider the following two cases.

1. Assume that the initial state s_{init} is in a recurrent class $R_{\bar{\pi}}^{\otimes i}$ for some $i \in \{1, \dots, h\}$. For any accepting set $\bar{F}_j^{\otimes}$, $\delta_{\bar{\pi}}^{\otimes i} \cap \bar{F}_j^{\otimes} \neq \emptyset$ holds by the definition of $\bar{\pi}$. The expected discounted reward for s_{init} is given by

$$V^{\bar{\pi}}(s_{init}) = \sum_{k=0}^{\infty} \sum_{s \in R_{\bar{\pi}}^{\otimes i}} \gamma^k p^k(s_{init}, s) \sum_{s' \in R_{\bar{\pi}}^{\otimes i}} P_{\bar{\pi}}^{\otimes}(s' \mid s) \mathcal{R}(s, a, s'),$$

where the action a is selected by $\bar{\pi}$. Since s_{init} is in $R_{\bar{\pi}}^{\otimes i}$, there exists a positive number $\bar{k} = \min\{k \ ; \ k \geq n, p^k(s_{init}, s_{init}) > 0\}$ [18]. We consider the worst scenario in this case. It holds that

$$\begin{aligned} V^{\bar{\pi}}(s_{init}) &\geq \sum_{k=n}^{\infty} p^k(s_{init}, s_{init}) \sum_{i=1}^{n} \gamma^{k-i} r_p \\ &\geq \sum_{k=1}^{\infty} p^{k\bar{k}}(s_{init}, s_{init}) \sum_{i=0}^{n-1} \gamma^{k\bar{k}-i} r_p \\ &> r_p \sum_{k=1}^{\infty} \gamma^{k\bar{k}} p^{k\bar{k}}(s_{init}, s_{init}), \end{aligned}$$

whereas all states in $R(MC_{\bar{\pi}}^{\otimes})$ are positive recurrent because $|S^{\otimes}| < \infty$ [19]. Obviously, $p^{k\bar{k}}(s_{init}, s_{init}) \geq (p^{\bar{k}}(s_{init}, s_{init}))^k > 0$ holds for any $k \in (0, \infty)$ by the Chapman-Kolmogorov equation [18]. Furthermore, we have $\lim_{k \to \infty} p^{k\bar{k}}(s_{init}, s_{init}) > 0$ by the property of irreducibility and positive recurrence [20]. Hence, there exists $\bar{p}$ such that $0 < \bar{p} < p^{k\bar{k}}(s_{init}, s_{init})$ for any $k \in (0, \infty]$ and we have

$$V^{\bar{\pi}}(s_{init}) > r_p \bar{p} \frac{\gamma^{\bar{k}}}{1 - \gamma^{\bar{k}}}.$$

Therefore, for any $\bar{m} \in (V^{\pi^*}(s_{init}), \infty)$ and any $r_p < \infty$, there exists $\gamma^* < 1$ such that $\gamma > \gamma^*$ implies $V^{\bar{\pi}}(s_{init}) > r_p \bar{p} \frac{\gamma^{\bar{k}}}{1-\gamma^{\bar{k}}} > \bar{m}$.

2. Assume that the initial state s_{init} is in the set of transient states $T_{\bar{\pi}}^{\otimes}$. $P_{\bar{\pi}}^{M^{\otimes}}(s_{init} \models \varphi) > 0$ holds by the definition of $\bar{\pi}$. For a recurrent class $R_{\bar{\pi}}^{\otimes i}$ such that $\delta_{\bar{\pi}}^{\otimes i} \cap \bar{F}_j^{\otimes} \neq \emptyset$ for each accepting set $\bar{F}_j^{\otimes}$, there exist a number $\bar{l} > 0$, a state $\hat{s}$ in $Post(T_{\bar{\pi}}^{\otimes}) \cap R_{\bar{\pi}}^{\otimes i}$, and a subset of transient states $\{s_1, \dots, s_{\bar{l}-1}\} \subset T_{\bar{\pi}}^{\otimes}$ such that $p(s_{init}, s_1) > 0$, $p(s_i, s_{i+1}) > 0$ for $i \in \{1, ..., \bar{l}-2\}$, and $p(s_{\bar{l}-1}, \hat{s}) > 0$ by the property of transient states. Hence, it holds that $p^{\bar{l}}(s_{init}, \hat{s}) > 0$ for the state $\hat{s}^{\otimes}$. Thus, by ignoring

rewards in $T^{\otimes}_{\bar{\pi}}$, we have

$$\begin{aligned}V^{\bar{\pi}}(s_{init}) &\geq \gamma^{\bar{l}} p^{\bar{l}}(s_{init}, \hat{s}) \sum_{k=0}^{\infty} \sum_{s' \in R^{\otimes i}_{\bar{\pi}}} \gamma^k p^k(\hat{s}, s') \sum_{s'' \in R^{\otimes i}_{\bar{\pi}}} P^{\otimes}_{\bar{\pi}}(s''|s') \mathcal{R}(s', a, s'') \\ &> \gamma^{\bar{l}} p^{\bar{l}}(s_{init}, \hat{s}) r_p \bar{p} \frac{\gamma^{\bar{k}'}}{1 - \gamma^{\bar{k}'}},\end{aligned}$$

where $\bar{k}' \geq n$ is a constant and $0 < \bar{p} < p^{k\bar{k}'}(\hat{s}, \hat{s})$ for any $k \in (0, \infty]$. Therefore, for any $r_p < \infty$ and any $\bar{m} \in (V^{\pi^*}(s_{init}), \infty)$, there exists $\gamma^* < 1$ such that $\gamma > \gamma^*$ implies $V^{\bar{\pi}}(s_{init}) > \gamma^{\bar{l}} p^{\bar{l}}(s_{init}, \hat{s}) \frac{r_p \bar{p} \gamma^{\bar{k}'}}{1-\gamma^{\bar{k}'}} > \bar{m}$.

The results contradict the optimality assumption of π^*. $\square$

Acknowledgment

First, I would like to express my deep sense of gratitude to my adviser Professor Toshimitsu Ushio, Graduate School of Engineering Science, Osaka University, for his invaluable, constructive advice and constant encouragement during this work. Professor Ushio's deep knowledge and his eye for detail have inspired me much.

I deeply appreciate Ami Sakakibara, Graduate school of Engineering Science, Osaka University, for fruitful comments and support.

Finally, I offer special gratitude to my family for their constant supports.

References

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

[2] E. M. Clarke, Jr., O. Grumberg, D. Kroening, D. Peled, and H. Veith, *Model Checking*, 2nd Edition. MIT Press, 2018.

[3] M. Kloetzer, C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Trans. Autom. Contr.*, vol. 53, no. 1, pp. 287–297, 2008.

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

[5] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *IEEE Trans. Autom. Contr.*, vol. 57, no. 11, pp. 2817–2830, 2012.

[6] A. Sakakibara and T. Ushio, "Decentralized supervision and coordination of concurrent discrete event systems under LTL constraints," in *Proc. 14th International Workshop on Discrete Event Systems*, 2018, pp. 18-23.

[7] C. Belta, B. Yordanov, and E. A. Gol, *Formal Methods for Discrete-Time Dynamical Systems.* Springer, 2017.

[8] M. L. Puterman, *Markov Decison Processes, Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., 1994.

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

[10] D. Sadigh, E. S. Kim, A. Coogan, S. S. Sastry, and S. Seshia, "A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications," *in Proc. 53rd IEEE Conference on Decision and Control*, pp. 1091-1096, 2014.

[11] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd Edition. MIT Press, 2018.

[12] M. Hiromoto and T. Ushio, "Learning an optimal control policy for a Markov decision process under linear temporal logic specifications," in *Proc. 2015 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, 2015, pp. 548-555.

[13] S. Sickert, J. Esparaza, S. Jaax, and J. Křetìnský, "Limit-deterministic Büchi au-

tomata for linear temporal logic," in *International Conference on Computer Aided Verification*, 2016, pp. 312-332.

[14] M. Hasanbeig, A. Abate, and D. Kroening, "Logically-constrained reinforcement learning," *arXiv:1801.08099v8*, Feb. 2019.

[15] E. M. Hahn, M. Perez, S. Schewe, F. Somenzi, A. Triverdi, and D. Wojtczak, "Omega-regular objective in model-free reinforcement learning," *Lecture Notes in Computer Science*, no. 11427, pp. 395–412, 2019.

[16] M. Hasanbeig, Y. Kantaros, A. Abate, D. Kroening, G. J. Pappas, and I. Lee, "Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantee," *arXiv:1909.05304v1*, 2019.

[17] A. K. Bozkurt, Y. Wang, M. Zavlanos, and M. Pajic, "Control synthesis from linear temporal logic specifications using model-free reinforcement learning," *arXiv:1909.07299*, 2019.

[18] R. Durrett, *Essentials of Stochastic Processes*, 2nd Edition. ser. Springer texts in statistics. New York; London; Springer, 2012.

[19] L. Breuer, "Introduction to Stochastic Processes", [Online]. Available: https://www.kent.ac.uk/smsas/personal/lb209/files/sp07.pdf

[20] S.M. Ross, *Stochastic Processes*, 2nd Edition. University of California, Wiley, 1995.

[21] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesv́ari, "Convergence results for single-step on-policy reinforcement learning algorithms" *Machine Learning*, vol. 38, no. 3, pp, 287–308, 1998.

[22] J. Kretínský, T. Meggendorfer, S. Sickert, "Owl: A library for ω-words, automata, and LTL," in *Proc. 16th International Symposium on Automated Technology for Verification and Analysis*, 2018, pp. 543–550.