

## 609 A Paths

610 An MDP begins in an initial state sampled from the initial state distribution  $s_0 \sim p_0(\cdot)$  and evolves  
 611 by transitioning from a current state  $s$  to a next state  $s'$  through an action  $a$ , as determined by the  
 612 transition function:  $s' = f(s, a)$ . In each state  $s$ , the policy observes a set of atomic propositions  
 613 provided by the labeling function  $L(s)$ . The sequence of visited states is called a path and is formally  
 614 defined below:

615 **Definition 3.** A path of an MDP  $M$  is defined as an infinite sequence of states  $\sigma = s_0 s_1 \dots$ ,  
 616 where each  $s_i \in S$ , such that  $p_0(s_0) > 0$  and for every  $t > 0$ , there exists an action  $a_t \in A$   
 617 with  $f(s_t, a_t) = s_{t+1}$ . We denote the  $t$ -th state in the sequence as  $\sigma[t]$ , the prefix up to  $t$  as  
 618  $\sigma[:t] = s_0 s_1 \dots s_t$ , and the suffix starting from  $t + 1$  as  $\sigma[t+1:] = s_{t+1} s_{t+2} \dots$ . The corresponding  
 619 sequence of labels for  $\sigma$  is referred to as the trace, defined by  $L(\sigma) := L(s_0)L(s_1) \dots$ .

## 620 B $\omega$ -Automata

621 An LTL formula  $\varphi$  can be translated into a finite-state automaton that operates over infinite paths,  
 622 known as an  $\omega$ -automaton. We denote the automaton corresponding to a specific formula  $\varphi$  by  $\mathcal{A}_\varphi$ .  
 623 An  $\omega$ -automaton  $\mathcal{A}_\varphi$  accepts a path  $\sigma$  if and only if  $\sigma \models \varphi$ , and rejects it otherwise. We begin by  
 624 formally introducing a general type of  $\omega$ -automaton, called a nondeterministic Rabin automaton  
 625 (NRA), which can be systematically derived from any LTL formula [78]. We then focus on specific  
 626 subclasses of NRAs that are particularly relevant to our work.

627 **Definition 4.** A nondeterministic Rabin automaton (NRA) derived from an LTL formula  $\varphi$  is defined  
 628 as a tuple  $\mathcal{A}_\varphi = (Q, q_0, \Sigma, \delta, \text{Acc})$ , where:

- 629 •  $Q$  is a finite set of automaton states;
- 630 •  $q_0 \in Q$  is the initial automaton state;
- 631 •  $\Sigma = 2^A$  is the input alphabet, where  $A$  is the set of atomic propositions;
- 632 •  $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$  is the transition function, which is complete and deterministic on  $\Sigma$  (i.e.,  
 633  $|\delta(q, w)| = 1$  for any  $q \in Q$  and  $w \in \Sigma$ ), but may include nondeterministic  $\varepsilon$ -transitions (i.e., it is  
 634 possible that  $|\delta(q, \varepsilon)| = 0$  or  $|\delta(q, \varepsilon)| > 1$ );
- 635 •  $\text{Acc}$  is a set of  $k$  accepting pairs  $\{(B_i, C_i)\}_{i=1}^k$  where  $B_i, C_i \subseteq Q$  for  $i \in \{1, \dots, k\}$ .

636 Given an infinite word  $\omega = w_0 w_1 \dots$ , a run of the automaton is a sequence of transitions  $\tau_\omega =$   
 637  $(q_0, w_0, q_1), (q_1, w_1, q_2), \dots$  such that  $q_{t+1} \in \delta(q_t, w_t)$  for all  $t \geq 0$ . The word  $\omega$  is *accepted* if  
 638 there exists such a run and at least one accepting pair  $(B_i, C_i)$  satisfying the Rabin condition: the  
 639 run visits some states in  $B_i$  infinitely often and all states in  $C_i$  only finitely often. This acceptance  
 640 condition is known as the Rabin condition, which can be formalized as  $\omega \models \varphi \iff \exists t : \text{Inf}(\tau_\omega) \cap B_i \neq \emptyset \wedge \text{Inf}(\tau_\omega) \cap C_i = \emptyset$  where  $\text{Inf}(\tau_\omega)$  denotes the set of states visited infinitely many  
 641 times in the run  $\tau_\omega$ . Similarly, a path  $\sigma$  is considered *accepted* by the automaton if its trace  $L(\sigma)$   
 642 forms a word accepted by  $\mathcal{A}_\varphi$ .  
 643

644 We focus on a specific subclass of NRAs known as limit-deterministic Büchi automata (LDBAs),  
 645 which feature a simplified acceptance criterion. Despite their reduced complexity, LDBAs retain  
 646 the full expressive power of general NRAs and can be systematically derived from LTL formulas  
 647 [79, 85].

648 **Definition 5.** An LDA is an NRA defined as  $\mathcal{A}_\varphi = (Q, q_0, \Sigma, \delta, \text{Acc})$  that satisfies the following:

- 649 • The acceptance condition consists of a single pair with an empty second set, i.e.,  $\text{Acc} = (B, \emptyset)$ ,  
 650 meaning that the run must visit some states in  $B$  infinitely often. This is known as the Büchi  
 651 condition, and for simplicity, we denote this acceptance condition by the set of accepting automaton  
 652 states itself, i.e.,  $\text{Acc} = B$ .
- 653 • The state space  $Q$  is partitioned into two disjoint subsets: an initial component  $Q_I$  and an accepting  
 654 component  $Q_A$ , satisfying:
  - 655 – All accepting states are contained in  $Q_A$ , i.e.,  $B \subseteq Q_A$ ;

- 656 – All transitions within  $Q_A$  are deterministic (no  $\varepsilon$ -transitions), i.e., for any  $q \in Q_A$ ,  $\delta(q, \varepsilon) = \emptyset$ ;
- 657 – Transitions cannot go from  $Q_A$  to  $Q_I$ , i.e., for all  $q \in Q_A$  and  $w \in \Sigma$ ,  $\delta(q, w) \subseteq Q_A$ .

658 The defining feature of LDBAs is that accepting runs must eventually enter the accepting compo-  
 659 nent  $Q_A$  and remain there permanently. Once this transition occurs, all subsequent behavior is  
 660 deterministic—this property is known as limit-determinism. LDBAs can be constructed such that  
 661 every  $\varepsilon$ -transition leads directly into  $Q_A$ , which ensures at most one such transition occurs along  
 662 any execution path. This construction, combined with limit-determinism, makes LDBAs particularly  
 663 well-suited for quantitative model checking in MDPs [79] unlike general nondeterministic automata.  
 664 Thus, we assume all LDBAs under consideration possess this structure.

## 665 C LTL Examples

666 LTL can be used to specify temporal properties of a wide range of robotics tasks:

- 667 • safety; e.g., “always avoid obstacles and remain below the joint angle and velocity thresholds”:

$$\varphi = \Box \left( \bigwedge_{\text{obstacle}_i} \text{dist\_to\_obstacle}_i > 0 \bigwedge_{\text{joint\_angle}_i} \text{joint\_angle}_i < \beta_{\max} \bigwedge_{\text{joint\_vel}_i} \text{joint\_vel}_i < \dot{\beta}_{\max} \right); \quad (13)$$

- 668 • reachability; e.g., “accelerate to a target velocity in x direction”:

$$\varphi = \Diamond \text{torso\_vel}_x > v_{\text{target}}; \quad (14)$$

- 669 • sequencing; e.g., “move to the position 1, then move to the position 2, and after that move to the  
 670 position 3”:

$$\varphi = \Diamond \left( \text{dist\_to\_pos}_1 < \delta_{\text{tol}} \wedge \Diamond \left( \text{dist\_to\_pos}_2 < \delta_{\text{tol}} \wedge \Diamond \left( \text{dist\_to\_pos}_3 < \delta_{\text{tol}} \right) \right) \right); \quad (15)$$

- 671 • repetition; e.g., “repeatedly monitor the region 1 and the region 2”:

$$\varphi = \Box \Diamond \text{dist\_to\_region}_1 < \delta_{\text{tol}} \wedge \Box \Diamond \left( \text{dist\_to\_region}_2 < \delta_{\text{tol}} \right). \quad (16)$$

## 672 D Finite-Memory Policies

673 **Definition 6.** A finite-memory policy for an MDP  $M$  is defined as a tuple  $\pi = (\mathfrak{M}, \mathfrak{m}_0, \mathfrak{T}, \mathfrak{a})$ , where:

- 674 •  $\mathfrak{M}$  is a finite set of modes (memory states);
- 675 •  $\mathfrak{m}_0 \in \mathfrak{M}$  is the initial mode;
- 676 •  $\mathfrak{T} : \mathfrak{M} \times S \times \mathfrak{M} \rightarrow [0, 1]$  is a probabilistic mode transition function such that for any current mode  
 677  $\mathfrak{m}$  and state  $s$ , the probabilities over next modes sum to 1, i.e.,  $\sum_{\mathfrak{m}' \in \mathfrak{M}} \mathfrak{T}(\mathfrak{m}' \mid \mathfrak{m}, s) = 1$ ;
- 678 •  $\mathfrak{a} : \mathfrak{M} \times S \times A \rightarrow [0, 1]$  is a probabilistic action selection function that assigns a probability to  
 679 each action  $a$  given the current mode  $\mathfrak{m} \in \mathfrak{M}$  and state  $s \in S$ .

680 A finite-memory policy acts as a finite-state machine that updates its internal mode (memory state)  
 681 as states are observed, and specifies a distribution over actions based on both the current state and  
 682 mode. The action at each step is thus selected according not only the current state but also the  
 683 current memory state of the policy. In contrast to standard definitions of finite-memory policies (e.g.,  
 684 [78, 80]), which typically assume deterministic mode transitions, this definition permits probabilistic  
 685 transitions between modes.

## 686 E Reinforcement Learning Objective

687 The objective of policy gradient algorithms for a given horizon  $H$  is to find the optimal policy  
 688 parameters  $\theta^* := \arg\max_{\theta} J(\theta)$ , where

$$J(\theta) := \mathbb{E}_{\sigma \sim M_{\pi_{\theta}}} [G_H(\sigma)]. \quad (17)$$

689 If the model is available and differentiable, one can utilize the following first-order gradients for  
690 optimization:

$$\nabla_{\theta}^{[1]} J(\theta) := \mathbb{E}_{\sigma \sim M_{\pi_{\theta}}} [\nabla_{\theta} G_H(\sigma)], \quad (18)$$

691 Alternatively, via the policy gradient theorem, one can employ the following model-free zeroth-order  
692 policy gradients

$$\nabla_{\theta}^{[0]} J(\theta) := \mathbb{E}_{\sigma \sim M_{\pi_{\theta}}} \left[ G_H(\sigma) \sum_{t=0}^{H-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]. \quad (19)$$

693 Both gradient estimates can be approximated using Monte Carlo sampling:

$$\bar{\nabla}_{\theta}^{[1]} = \frac{1}{N} \sum_{i=1}^N \nabla_{\theta} G_H(\sigma_i), \quad \sigma_i \sim M_{\pi_{\theta}}; \quad (20)$$

$$\bar{\nabla}_{\theta}^{[0]} = \frac{1}{N} \sum_{i=1}^N G_H(\sigma_i) \sum_{t=0}^{H-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t), \quad \sigma_i \sim M_{\pi_{\theta}}. \quad (21)$$

## 694 F Parking Example: Differentiable RL vs Non-Differentiable RL

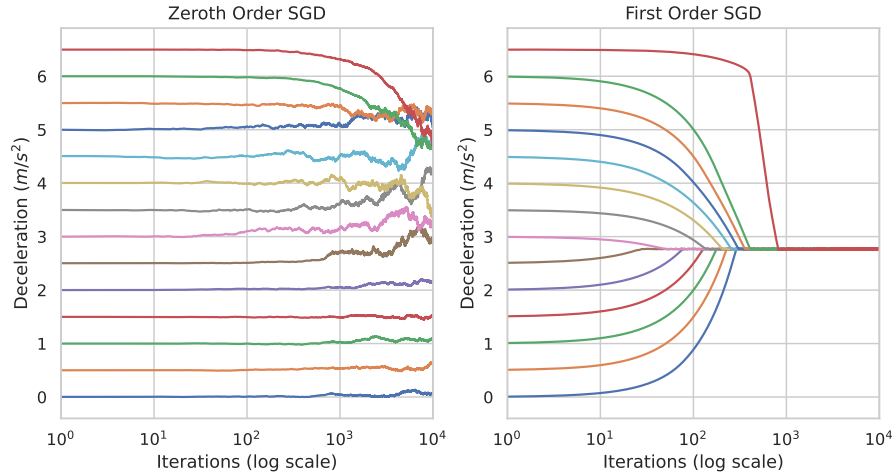


Figure 5: Convergence speed comparison of stochastic gradient descent algorithms using  $\bar{\nabla}_{\theta}^{[0]}$  and  $\bar{\nabla}_{\theta}^{[1]}$  for the parking example ( $N = 10$ ).

## 695 G LTL Specifications and Automata used in Experiments

### 696 • CartPole:

697  $G(\text{"position\_x"} > -10" \ \& \ \text{"position\_x"} < 10") \ \& \ G(\text{"velocity\_x"} > -10.0" \ \& \ \text{"velocity\_x"} < 10.0")$   
698  $\ \& \ F(\text{"cos\_theta"} < -0.5" \ \& \ F(\text{"cos\_theta"} > 0.5"))$   
699

### 700 • Hopper:

701  $G(\text{"torso\_height"} > -11.0" \ \& \ GF(\text{"torso\_height"} > -10.5" \ \& \ F(\text{"torso\_velocity\_x"} > 1.0" \ \& \ F(\text{"torso\_velocity\_x"} < 0"))$   
702

### 703 • Cheetah:

704  $G(\text{"tip\_height"} > -7.5" \ \& \ GF(\text{"tip\_height"} > -7.0" \ \& \ F(\text{"tip\_velocity\_x"} > 3.0" \ \& \ F(\text{"tip\_velocity\_x"} < 0"))$   
705

### 706 • Ant:

707  $G(\text{"torso\_height"} > 0.0" \ \& \ GF(\text{"torso\_height"} > 0.5" \ \& \ F(\text{"torso\_velocity\_x"} > 1.5" \ \& \ F(\text{"torso\_velocity\_x"} < 0"))$   
708



Figure 6: The  $\omega$ -automaton derived from  $\varphi_{\text{cartpole}}$  from (9).

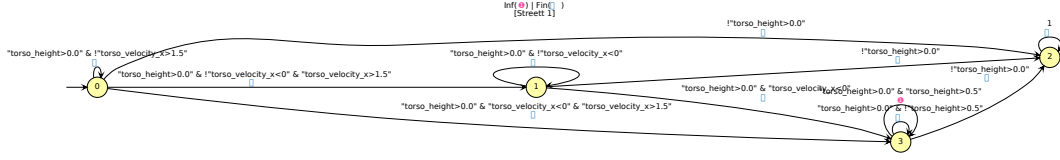


Figure 7: The  $\omega$ -automaton derived from  $\varphi_{\text{legged}}$  from (10) for the Ant environment.