Formal Language Constraints for Markov Decision Processes

Eleanor Quint[*†], **Dong Xu**[‡], **Haluk Dogan**[†], **Zeynep Hakguder**[†], **Stephen Scott**[†], **Matthew Dwyer**[‡]
University of Nebraska-Lincoln[†], University of Virginia[‡]

Abstract

In order to satisfy safety conditions, a reinforcement learned (RL) agent may be constrained from acting freely, e.g., to prevent trajectories that might cause unwanted behavior or physical damage in a robot. We propose a general framework for augmenting a Markov decision process (MDP) with constraints that are described in formal languages over sequences of MDP states and agent actions. Constraint enforcement is implemented by filtering the allowed action set or by applying potential-based reward shaping to implement hard and soft constraint enforcement, respectively. We instantiate this framework using deterministic finite automata to encode constraints and propose methods of augmenting MDP observations with the state of the constraint automaton for learning. We empirically evaluate these methods with a variety of constraints by training Deep Q Networks in Atari games as well as Proximal Policy Optimization in MuJoCo environments. We experimentally find that our approaches are effective in significantly reducing or eliminating constraint violations with either minimal negative or, depending on the constraint, a clear positive impact on final performance.

1 Introduction

The ability to impose safety constraints on an agent is a key requirement for the deployment of reinforcement learning (RL) systems to unconstrained real-world environments [4]. Controllers that are derived mathematically typically rely on agent behavior remaining within a pre-defined "envelope of safety" and fully *a priori* analysis in order to guarantee safe operation [5]. This approach limits controllers to trajectories that are sufficiently simple to remain within pre-defined operational limits. Alternatively, reinforcement learned controllers are free to learn control trajectories that better suit their tasks and goals, but verifying their safety properties can be difficult and there is no obvious method of constraining the learning process to ensure it does not result in an unsafe controller. A variety of existing approaches to safety exist, but almost all focus on modifying the agent's optimization or exploration to be safer [19]. These safety advances have been separate from much of the rapid progress that has been made in deep reinforcement learning. In contrast, we ask: What if we could analyze the safety properties of an environment itself *a priori*, entirely agnostic to the properties of the learning algorithm or learned agent? By wrapping a base environment in constraints, the set of possible solutions to the learning problem can be modified, guiding a general-purpose RL algorithm to learn to avoid safety constraint violations and still maximize performance in the base environment.

Rather than trying to specify properties of the controller and then verify its safety, we introduce a general *a priori* method of adding constraints, specified in a formal language, to the Markov decision process itself. These constraints can be hard or soft, depending on the criticality of the constraint, and can be defined on any sequence of MDP states and agent actions for which a formal language

[*]Correspondence to equint4@huskers.unl.edu

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

can be specified. Further, these constraints can be used with any reinforcement learning system that operates in the Markov decision process setting.

A significant advantage of specifying constraints using formal languages is that they already form a well-developed basis for engineering components of safety-critical systems [26, 14, 32, 6] and we can exploit those constraints further for safe reinforcement learning. Moreover, the recognition problem for many classes of formal language imposes modest computational requirements which has made them suitable for efficient runtime verification [13] and which we exploit to incorporate them into training and deployment of MDPs.

Our constraints have two varieties: **hard constraints,** used when a safety condition must not be violated for any reason, and **soft constraints,** in the case that violations are discouraged, but are allowed if it leads to great enough reward. We employ separate mechanisms in each case to train an agent that respects each type of constraint without unduly negatively impacting performance.

Our main contribution is the introduction of a method of applying formal language constraints to learning Markov decision processes over MDP state and action sequences. We discuss a variety of methods for augmenting MDP state, discuss strategies for reward shaping with soft constraints, and how hard constraints are accommodated through **action shaping**, which limits action choices during training to those that meet constraints. We experimentally validate these methods by applying example constraints, based on action sequences and/or MDP states, to Atari and MuJoCo environments and then report on learning, returns, and constraint violations made by standard model-free deep RL algorithms.

The remainder of this work is organized as follows. Section 2 describes our formal language constraint framework. Section 3 presents our experimental setup and results. Finally, we present related work in Section 4 and future work in Section 5.

2 Formal Language Constraint Framework

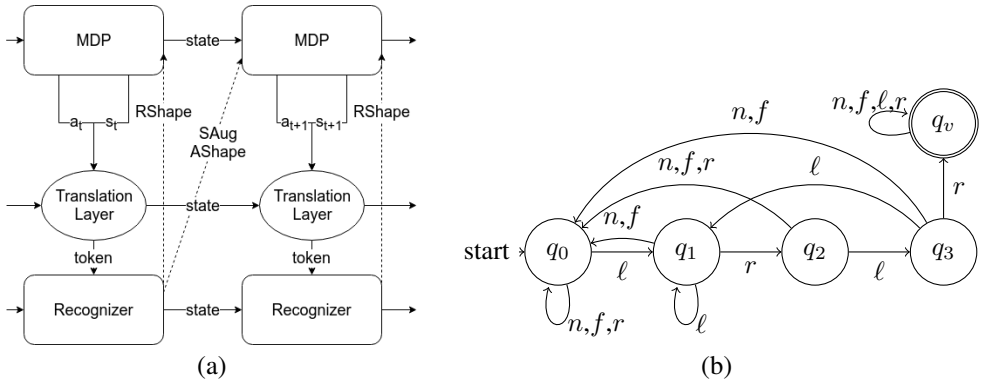
A formal language constraint $C \subset (S \times A)^*$ is defined as a set of prohibited trajectories, sequences of states and actions over $S \times A$. We call an MDP augmented by formal language constraints a "Constraint Augmented MDP" (CAMDP), defined by a 6-tuple $M' = (M, D, f_{tl}, SAug, RShape, AShape)$, where M is the base MDP and D is the set of **recognizers** that encode the constraint languages whose accepting states imply a constraint violation. f_{tl} is a set of **translation functions** (one for each recognizer in D) that is evaluated at each MDP time step and outputs a token to be input into the associated recognizer in D. Translation functions implicitly operate on prefixes of trajectories through the MDP state space and abstract the action and MDP state to reflect equivalence classes that are discriminated by recognizers. The last three elements are functions of the recognizer state: $SAug : D \times S \to S'$ is a function that augments the MDP state with information about the recognizer state, $RShape : D \times \mathbb{R} \to \mathbb{R}$ performs reward shaping in the case of a soft constraint, and $AShape : D \times A \to A'$ restricts the available actions to prevent constraint violations in the case of a hard constraint.

This framework interacts with the MDP in a simple loop (Figure 1(a)). At time step t, the MDP state and agent's selected action are passed to the translation layer, f_{tl}, and the output token is fed into the recognizer, D. Then, the recognizer's state is used to compute reward shaping for the return with $RShape$ at time t, and the allowed action set with $AShape$ and state augmentation with $SAug$ for time step $t + 1$.

2.1 Recognizer

D includes a recognizer for each constraint. Using multiple constraints simplifies their expression by allowing them to focus on individual components of the state or action space (e.g., a constraint on the behaviour of each joint). The framework's only major assumption about the recognizers is that they define some meaningful state that may be used as input to $SAug$, $AShape$, and $RShape$.

Our implementation of the framework uses a deterministic finite automaton (DFA) as the recognizer for each constraint, defined as $(Q, \Sigma, \delta, q_0, F)$, where Q is the finite set of the DFA's states, Σ is the alphabet over which the constraints are defined, $\delta : Q \times A \to Q$ is the DFA's transition function, $q_0 \in Q$ is the DFA's start state, and $F \subset Q$ is the set of accepting states which represent constraint



(a) (b)

Figure 1: (a) Illustration of the formal language constraint framework operating through time. In the same way state is carried forward through time by the MDP, it's also propagated by the translation layer, f_{tl}, and the recognizer, D. (b) No-1D-dithering constraint: $.^* (\ell\, r)^2$

violations. The DFA is advanced at each time step with the output of the translation layer and, after reaching an accepting state, is reset to the initial state.

Although we've chosen to use DFAs as a relatively simple recognizer, the other parts of our instantiation of the framework can be easily modified to work with automata which encode richer formal languages, e.g., pushdown automata or hybrid automata. Further, real-valued or continuous-time actions and states may be discretized with f_{tl} in order to be used with the formal language framework. Restricting constraints to being encoded in discrete, finite languages may be awkward for some classes of constraint, but this can be ameliorated by a more sophisticated translation function.

2.2 Translation Layer

Formal languages are straightforward to define over discrete, finite sets of actions and MDP states, but less so when working with continuous or infinite spaces. The translation layer can be any function that accepts inputs of the MDP state and agent action at each time step, and outputs a token in the discrete, finite language of the associated recognizer.

Our implementation of the framework uses the following translation functions. In **Atari**, the output token is the passed-through action input, which we chose because each action set is small and we only define constraints on action sequences in the Atari environments. In **MuJoCo**, since the constraints are defined per joint, the translation layer discretizes the joint torques by sign or by magnitude and assigns all values beyond a pre-defined range to a single token to make the space finite. The Reacher environment uses a similar state space discretization of the distance from the goal state.

2.3 Hard Constraints and Action Shaping

When safety constraints are strict, the set of available actions in the MDP is modified to ensure a constraint violation cannot occur. $AShape$ is a function of the automaton state that removes actions from the available action set that, if executed, would move the recognizer into an accepting state. If the constraint is not strict, $AShape$ can be the identity function. Because this enforcement mechanism requires knowledge of which actions will or will not set the agent and MDP on a violating trajectory, $AShape$ may only be applied when the dependence of violations on MDP state is well-defined. For example, when violations are independent of state as in purely action based constraints.

Our implementation of the framework initially allows the agent to freely choose its action, but before finalizing that choice, simulates stepping the DFA with the resulting token from the translation layer and, if that lookahead action choice would move it into a violating state, it moves to the next best choice until a non-violating action is found. If none is found (which is impossible for the constraints we evaluate), the episode is terminated immediately. In our experiments, we refer to this approach as **constraint enforcement** (ce). The application of ce during training is called **action shaping** (in contrast with reward shaping for training policies with soft constraints). Applying ce during

deployment, or testing, **guarantees** the lack of constraint violations even if the trained MDP may exhibit them.

2.4 Soft Constraints and Reward Shaping

Reward shaping is used at training time to implement soft constraints. One of two mechanisms may be employed. The first is *sparse* reward shaping, in which the MDP reward signal is only modified when the constraint is violated. The second is *dense* reward shaping, which applies shaping at every time step according to the function $RShape(r_t, d_{t-1}, d_t) = r + \gamma\Phi(d_t) - \Phi(d_{t-1})$, where d_t is the recognizer state at time t and Φ is a function that decreases as the recognizer state moves closer to a violation. This has the general form of a potential-based shaping function, which facilitates learning by acting as a dense feedback signal without changing the optimality or near-optimality of policies learned in the CAMDP M' relative to the MDP M [40]. However, the optimal policy in the CAMDP should trade off minimizing violations and maximizing reward, so we modify this reward shaping scheme such that, after a constraint violation, no reward is re-gained from the recognizer being reset. This results in a mechanism we call *pseudo-potential based shaping*, which modifies the optimal policy to reflect the constraints.

One implementation of the potential function Φ is the product of the *a priori* selected value of violating the constraint and an estimate of the probability the constraint will be violated by the end of the episode. We estimate Φ with two counters attached to each automaton state: $\Phi(d_t) = r_c * \frac{violations(d_t)}{visits(d_t)}$, where r_c is the value assigned to the constraint, $violations$ counts the number of times the constraint has been violated after visiting d_t, and $visits$ counts the total number of times d_t has been visited. This allows reward shaping to dynamically adjust the value of each DFA state according to how close the agent is to a constraint violation, but are updated only after each episode to ensure Φ is stationary.

2.5 State Augmentation

As part of the framework, we propose augmenting MDP state observations that the agent uses to choose actions with the state of the recognizer. This allows the agent to learn how to avoid constraint violations as part of learning how to maximize return. We describe a variety of approaches to $SAug$ in Section 3.1, how they were applied in each Atari and MuJoCo environment in Section 3.3, and find that each of these methods is appropriate in a different setting in Section 3.4.

2.6 Algorithm

Algorithm 1 implements the basic reinforcement learning training loop with a Constraint Augmented MDP. Implementing $AShape$ can be done via constraint enforcement by simply simulating the chosen action in D and checking for a violation, requesting the agent's second-choice action if there is one. Thus, the increase in time complexity of our algorithm over non-constrained RL depends on this and on the time required to compute f_{tl}, $SAug$, and $RShape$.

Input: CAMDP M' , policy π
Reset MDP M to get initial state s_0
Reset recognizer D to get initial state d_0
repeat
  Filter available action set to get $A'_t = AShape(d_{t-1})$
  Choose action from available set with $a_t = \text{argmax}_{a \in A'_t} \pi(a \mid s_{t-1})$
  Advance MDP $s_t, r_t = M(a_t)$
  Update D with $d_t = D(f_{tl}(s_t, a_t))$
  Augment state $s_t = SAug(s, D)$
  Shape reward $r_t = RShape(r, D)$
  Train agent with modified reward and state
until episode terminates

Algorithm 1: Applying formal language constraints to RL training loop over an episode

3 Experimental Evaluation

We present experiments aimed at evaluating instantiations of the proposed framework that help to answer the following research questions:
(**RQ1**) What methods are most effective in minimizing constraint violations and achieving high reward?
(**RQ2**) Can final performance in the MDP be enhanced by simple constraints?

We address these questions with experiments using the following state augmentations, constraints, and environments.

3.1 State Augmentation Methods

We developed three variants of the framework's $SAug$ function. **One-hot DFA state encoding** concatenates a one-hot representation of the DFA state at time t to the state representation. **Action history** concatenates the last k agent actions from time $t-k$ to time t in a one-hot encoding to the state representation (in our experiments, we set $k = 10$). Finally, in **DFA state embedding**, we augment with an embedding of the DFA state learned *a priori* by node2vec [24]. We directly applied node2vec's standard random walk with $d = 3$, and performed 200 iterations of 200 random walks of each DFA, each of length 80. Our context window size was 5, and our random walks were a combination of BFS and DFS, setting $p = q = 1$.

3.2 Constraints

We evaluated our implementation of the framework on two families of constraints.

No-dithering: A no-dithering constraint prohibits movements in small, tight patterns that cover very small areas. For Atari environments, in one dimension, we define **no-1D-dithering** as having a violation as $.^* (\ell\, r)^2$, i.e., never move left then right then left then right; here "$.^*$" is any sequence over actions from A. The automaton encoding this constraint is depicted in Figure 1(b). In environments with two dimensional action spaces, such as Atari Seaquest, we generalize this to **no-2D-dithering**, which extends to vertical and diagonal moves and constrains actions that take the agent back to where it started in at most four steps.[2] In MuJoCo environments, the constraint is applied per joint and the translation layer maps negative and positive-valued actions to 'ℓ' and 'r', respectively.

No-overactuating: In general, a no-overactuating constraint prohibits repeated movements in the same direction over a long period of time. In Atari environments, in one dimension, a violation is $.^* (\ell^4 \cup r^4)$, i.e., never move left four times in a row or right four times in a row. In two dimensions, this is extended to include vertical move groups: $.^* (L^4 \cup R^4 \cup U^4 \cup D^4)$. Each of left ($L$), right ($R$), up ($U$) and down ($D$) groups contains the primary direction it's named after and diagonal moves that contain the primary direction, e.g., $L = \ell \cup \ell{+}u \cup \ell{+}d$, where "$\ell{+}u$" is the atomic left-up diagonal action. In MuJoCo environments, overactuation is modelled as occurring when the sum of the magnitudes of joint actuations exceeds a threshold. This requires the translation layer to discretize the magnitude in order to set up a DFA that calculates an approximate sum. The MDP state-based version is called **dynamic actuation**, which sets the constraint threshold dynamically based on a discretized distance from a goal state.

3.3 Environments and Experiments

We evaluated the framework on these augmentations and constraints in three Atari environments and two MuJoCo environments.[3]

3.3.1 Atari

We focused on three Atari games: Breakout, Space Invaders, and Seaquest [8], which have diverse action spaces and high-reward game play strategies. For each game we considered the two generic constraints described in Section 3.2. Each constraint is applicable to each of the games. The Atari environments used no translation layer other than to pass the action directly as the token to the recognizer DFA. Deep Q networks were used for action selection and identical in architecture to [37] except for concatenating the output a learned linear function of $SAug$ to the output of the final convolutional layer. There are two main sets of Atari experiments, a sparse soft constraint with varied[4] reward shaping per constraint violation (Table 1), and a hard constraint (Table 2). In the hard constraint experiments, non-baseline approaches were trained via action shaping, where an action

[2] The regex describing this constraint is 7320 characters long and is thus omitted.

[3] Code can be found at `https://github.com/neurips2019submission/Formal-Language-Constraints-for-Markov-Decision-Processes`

[4] We tested shaping penalty values $r_c \in \{-1, -10, -100, -1000\}$. Results for $r_c = -10$ are reported in Table 1; other results are in supplementary material.

Table 1: Per-episode rewards for Atari environments and soft constraints with reward shaping $r_c = -10$

Environments	Constraints	Baseline		Reward Shaping (Violation Penalty $r_c = -10$)							
				Shaping Only		State Augmentation					
						action history		dfa one-hot		dfa embedding	
		rewards	violations	rewards	violations	rewards	violations	rewards	violations	rewards	violations
Breakout	Actuation	**27.06 ± 12.38**	60.82 ± 336.80	24.23 ± 12.88	4.85 ± 36.28	17.86 ± 9.11	4.55 ± 49.06	21.27 ± 9.59	**0.23 ± 0.48**	19.36 ± 8.98	2.97 ± 106.78
	Dithering	**27.97 ± 13.09**	68.37 ± 532.01	25.54 ± 12.46	0.27 ± 0.50	26.99 ± 12.58	0.20 ± 0.43	26.45 ± 11.87	**0.15 ± 0.37**	27.45 ± 12.80	0.18 ± 0.41
Space Invader	Actuation	248.76 ± 124.82	99.36 ± 53.49	266.62 ± 105.53	16.81 ± 15.34	234.50 ± 122.84	6.18 ± 5.67	248.44 ± 123.18	**0.64 ± 0.91**	**308.48 ± 144.38**	1.08 ± 1.28
	Dithering	274.73 ± 175.72	3.39 ± 3.27	246.56 ± 144.70	2.18 ± 1.89	271.08 ± 182.96	0.76 ± 0.99	251.66 ± 137.83	**0.33 ± 0.59**	**309.72 ± 168.20**	**0.33 ± 0.59**
Seaquest	Actuation	940.43 ± 691.40	57.69 ± 37.79	1389.10 ± 845.49	14.56 ± 11.80	1589.61 ± 685.61	4.67 ± 3.71	1474.23 ± 680.18	**0.75 ± 1.24**	**1643.71 ± 697.86**	12.03 ± 9.95
	2d-dithering	1307.50 ± 936.24	6.13 ± 5.70	632.90 ± 803.84	3.45 ± 3.83	**1456.64 ± 689.19**	1.29 ± 1.30	1310.88 ± 874.82	**0.93 ± 1.10**	1032.56 ± 808.08	2.06 ± 1.90

chosen during training is tested for a constraint violation before applied to the environment. If a violation occurs, the action is changed via constraint enforcement (ce) to the agent's highest-scoring action choice that does not violate the constraint. All agents except Baseline were trained this way (action shaping only; no reward shaping). Table 2 presents results both where ce is applied and not applied during testing.

Our loss function and optimizer were mean absolute error and Adam with a learning rate of 2.5×10^{-4}. Our discount future reward $\gamma = 0.99$ and we used ϵ-greedy as an exploration policy with ϵ decreasing from 1 to 0.1 over 10M steps. We set the experience replay memory limit to 1M samples. Every state of the game was represented with 4 frames. We trained the network with random mini-batches of size 32 from the replay memory at every 4 steps. The target network was updated every 10K steps. For each approach of Section 3.1 and our baseline methods, we trained the agent for 10M steps for 10 different training seeds. We tested each of our 10 trained agents for 10 different test seeds. Each test case includes 100 game episodes running to completion.

3.3.2 MuJoCo

We ran another set of experiments on two MuJoCo environments: Reacher with the overactuation and MDP state-based dynamic overactuation constraints, and HalfCheetah with the dithering constraint [9]. The unmodified OpenAI Baselines [15] implementation of Proximal Policy Optimization [44] was used for action selection, except for state augmentation concatenated with the network input. The only state augmentation tested was dfa one-hot and the only tested constraint was the dense soft constraint discussed in Section 2.4 with reward shaping value $r_c \in \{0, -1, -10, -100, -1000\}$. Training took place over 1M steps and was repeated with 20 seeds. Final results were calculated as the mean of the 100 episodes with greatest total reward, as done in the OpenAI Baselines library.

Both constraints were applied to each joint, so there were two instances of the no-overactuating constraint applied to Reacher and six of the no-dithering constraint applied to HalfCheetah. The translation layer for Reacher calculated an approximate sum of the action magnitudes by discretizing each action in increments of 0.2 into 15 tokens (the first for 0 to 0.2, the second for 0.2 to 0.4, etc.) and assigning all action values with magnitude greater than 3 to a special token. Then, the threshold for violating the overactuation constraint was set to 0.8 over three time steps. A similar state space discretization was calculated using the Euclidean distance from the goal state for the dynamic actuation constraint. The translation layer for HalfCheetah discretized the space of actions into positive 'r' and negative 'ℓ' tokens, with 0 assigned arbitrarily to negative. Then, the same no-1D dithering constraint as in Atari is applied.

3.4 Results and Analysis

3.4.1 Research Question 1

Tables 1 and 2 present average numbers of rewards and violations per test episode. Every instance in Table 1 is trained with reward shaping for soft constraints and trained with action shaping for hard constraints in Table 2. In each table, "*baseline*" denotes performance of an agent trained without any state augmentation or (reward or action) shaping, i.e., the same approach as [37], and all other names are identical to those presented in Section 3.1. Each value is combined with an error term indicating one standard deviation, and a "$-$" means there were no violations due to constraint enforcement (ce) during testing. Boldface numbers on a gray background indicate the best values for that row.

When using reward shaping for soft constraints with a reward shaping penalty of -10, *dfa one-hot* always had the minimum number of violations, particularly over *shaping only* and *baseline*. Further,

Table 2: Per-episode rewards and violations for Atari environments and hard constraints trained with action shaping. Results are presented with and without hard constraint enforcement (*ce*) applied at test time.

Environments	Constraints	Baseline		Action Shaping							
				Shaping Only		State Augmentation					
						action history		dfa one-hot		dfa embedding	
		rewards	violations	rewards	violations	rewards	violations	rewards	violations	rewards	violations
Breakout	Actuation	**27.06 ± 12.38**	60.82 ± 336.80	22.06 ± 12.83	1484.88 ± 4932.09	20.66 ± 9.41	43.19 ± 194.62	22.82 ± 11.66	51.76 ± 602.77	22.36 ± 11.81	**21.58 ± 67.12**
	Actuation w/ce	22.73 ± 11.88	-	22.62 ± 12.69	-	22.43 ± 9.84	-	**24.90 ± 10.30**	-	24.72 ± 11.33	-
	Dithering	**27.97 ± 13.09**	68.37 ± 532.01	27.03 ± 11.54	1.77 ± 2.06	21.97 ± 8.71	16.36 ± 236.86	20.80 ± 12.22	**0.57 ± 9.63**	24.11 ± 12.28	10.54 ± 198.08
	Dithering w/ce	**27.71 ± 12.74**	-	25.93 ± 11.03	-	22.07 ± 8.89	-	20.88 ± 12.30	-	23.31 ± 11.26	-
Space Invader	Actuation	248.76 ± 124.82	99.36 ± 53.49	222.93 ± 112.38	103.86 ± 47.35	224.97 ± 127.06	64.07 ± 34.94	**256.70 ± 130.76**	**34.10 ± 18.01**	206.80 ± 117.66	64.21 ± 25.88
	Actuation w/ce	254.06 ± 147.98	-	254.41 ± 132.52	-	244.41 ± 137.48	-	**280.87 ± 164.63**	-	231.11 ± 129.75	-
	Dithering	274.73 ± 175.72	3.39 ± 3.27	250.62 ± 138.48	2.73 ± 2.18	**298.70 ± 169.49**	2.96 ± 2.81	280.42 ± 166.95	7.15 ± 7.51	218.02 ± 107.24	**1.88 ± 1.94**
	Dithering w/ce	223.81 ± 113.50	-	240.20 ± 127.94	-	**299.05 ± 169.60**	-	260.34 ± 145.54	-	217.96 ± 110.42	-
Seaquest	Actuation	940.43 ± 691.40	57.69 ± 37.79	980.31 ± 539.36	31.96 ± 18.48	1098.78 ± 532.28	42.10 ± 27.86	**1154.45 ± 753.72**	**25.44 ± 21.70**	900.82 ± 532.79	50.37 ± 39.90
	Actuation w/ce	982.71 ± 737.38	-	980.89 ± 551.03	-	1093.44 ± 520.48	-	**1199.64 ± 753.53**	-	953.22 ± 595.25	-
	2d-Dithering	1307.50 ± 936.24	**6.13 ± 5.70**	909.67 ± 471.31	10.40 ± 8.97	1394.83 ± 737.04	9.91 ± 7.43	**1462.38 ± 831.41**	19.24 ± 17.19	860.70 ± 550.56	7.13 ± 6.16
	2d-Dithering w/ce	1276.26 ± 917.37	-	998.90 ± 491.18	-	1443.19 ± 724.14	-	**1526.45 ± 845.42**	-	917.83 ± 572.07	-

Table 3: Mean per-episode MuJoCo rewards and violations with soft constraints. Top row displays the reward shaping value r_c.

Environment	Constraint	Baseline		Reward Shaping Value									
				0		−1		−10		−100		−1000	
		rewards	violations	rewards	violations	rewards	violations	rewards	violations	rewards	violations	rewards	violations
Half cheetah	dithering	1555.30 ± 27.42	82.84 ± 6.26	1458.68 ± 32.23	80.57 ± 5.74	2054.84 ± 451.78	73.06 ± 13.37	2524.10 ± 496.68	62.31 ± 11.25	1495.21 ± 165.21	43.27 ± 10.21	639.00 ± 30.38	16.73 ± 6.70
Reacher	actuation	−6.55 ± 0.94	0.61 ± 0.06	−6.28 ± 0.51	0.59 ± 0.04	−6.55 ± 0.98	0.02 ± 0.03	−5.28 ± 0.22	0.00 ± 0.00	−8.36 ± 0.40	0.00 ± 0.00	−13.44 ± 0.61	0.00 ± 0.00
	dynamic actuation		0.00 ± 0.00	−5.93 ± 1.67	0.00 ± 0.00	−5.69 ± 1.02	0.00 ± 0.00	−5.53 ± 1.32	0.00 ± 0.00	−4.75 ± 0.88	0.00 ± 0.00	−11.40 ± 0.61	0.00 ± 0.00

except for the Breakout environment, there was always a state-augmented approach (typically *dfa embedding*) with maximum reward. Table 3 shows a similar result for MuJoCo. Thus, we conclude that shaping plus state augmentation can effectively reduce constraint violations while yielding good reward performance.

When using action shaping for hard constraints, if no ce is used at test time, there was always at least one state-augmented approach that was better than *shaping only* in terms of violations, and, on all tests except Seaquest/2d-Dithering, better than *baseline*. Further, in environments other than Breakout, there was always a state-augmented method with better rewards as than both *shaping only* and *baseline*. With ce applied at test time, except for Breakout/Dithering, there was always at least one state-augmented approach with better rewards than *baseline* and *shaping only*. Thus, we conclude that the application of both state augmentation and action shaping can effectively reduce constraint violations while improving (or at least not significantly negatively impacting) total reward.

3.4.2 Research Question 2

Separate from the overall effect of constraints on rewards and violations, we are interested in whether the application of constraints can enhance performance in the MDP, regardless of constraint violations.

Both Tables 1 and 2 show that, while training in the presence of constraints in Breakout does not tend to improve rewards, training with hard or soft constraints with shaping and state augmentation clearly does boost rewards in Space Invader and Seaquest. The use of these constraints during training seems to encourage exploration of better policies, we speculate by limiting exploration of ineffective policies. For example, moving in a tight pattern and ending up where one started is often not a good policy for Seaquest game play. Since this is prohibited by no-2D-dithering, arguably this constraint is forcing the agent to explore alternatives, and hence discover better policies.

The constraints also clearly improved performance in the MuJoCo environments, seen in Table 3. Here, we examine different choices of reward shaping value and find that, in all cases, there is a medium value that is large enough to have an effect, but not so large that it radically changes the underlying reward function. The zero-valued reward shaping trials act as an ablation study where the augmentation is present, but without reward shaping. Results show that, although adding only augmentation produces mixed results (helpful for Reacher and unhelpful for HalfCheetah), using both reward shaping and augmentation clearly improves performance. Interestingly, performance is still enhanced over zero shaping when using the dynamic actuation constraint, even though no agent is violating the constraint by the end of training. We interpret this as evidence that the constraints influence the exploration process as training proceeds.

4 Related Work

Previous work in safety in reinforcement learning falls under two general categories; some work incorporates safety with a modified optimality criterion, while others modify the exploration process to avoid undesirable situations [19].

Optimality Criterion Broadly, these works change the optimization objective to incorporate safety in some way, including optimizing to maximize worst-case performance [20, 41, 45]. Others introduce a risk function and optimize the linear combination of risk and return [43, 22], and still others modify the criterion to optimize reward subject to constraints, often on the allowed variance of the return [38, 30, 12, 2]. Our model of enforcing soft constraints with reward shaping resembles these methods, though we modify the MDP directly rather than the learner's optimality criterion.

Exploration Process Here, external information is provided during exploration to enhance safety, including using prior knowledge to initialize training [16, 17] and the use of examples to learn a model for safe off policy-learning, instead of random exploration [1, 46]. Others harness a teacher, either a human or an automated controller, available during exploration for guidance [18, 23, 47, 42, 31, 48]. Our method uses external information during exploration and is most similar to the usage of external information [21, 33], where the value function is replaced by a risk-adjusted utility function. The risk function is learned during training which requires random exploration to discover which trajectories should be avoided. In contrast, our method uses constraints which are known *a priori* and can be specified exactly. In addition, we augment MDP states with the constraint recognizer state information to aid learning how to avoid constraint violations. The instantiation of our framework employs DFAs to represent undesirable action sequences.

Automata have been used in RL for task specification or as task abstractions (options) in hierarchical reinforcement learning [28, 35, 49, 25, 39]. In some cases, these automata were derived from Linear Temporal Logic (LTL) formulae, in others LTL or other formal language formulae have been directly used to specify tasks [27]. Littman et al. [36] defined a modified LTL to be used in reinforcement learning. In robotics, LTL is used for task learning [35], sometimes in conjunction with teacher demonstrations [34]. In contrast, our work uses automata to disallow action sequences for safety.

Teacher Advice and Reward Shaping A subset of reinforcement learning safety that uses external information during the exploration process uses the potential-based reward shaping mechanism [40]. Wiewiora et al. [50] introduce a general method for incorporating arbitrary advice into the reward structure of an RL agent. Camacho et al. [11, 10] use DFAs with static reward shaping attached to states to express non-Markovian rewards. We build on this with a learned reward shaping function in the case of dense soft constraints, and by considering the translation of actions and states into the symbols used in the DFA alphabet. Other work in verifying the safety of deep RL policies includes Bastani et al. [7], which learns a decision tree-based policy and verifies it with a scalable algorithm, as opposed to our method of modifying the MDP itself.

Similar to teacher advice is shielding [29, 3], in which an agent's actions are filtered through a shield, which blocks actions that would introduce an unsafe state (similar to our constraint enforcement for hard constraints). Their work also involves constructing shields via probabilistic model checking, involving learning a model of adversary behavior and partitioning the MDP into zones to delineate safe from unsafe situations. They do not explore action shaping or state augmentation, which we found to be beneficial in achieving high reward performance and constraint conformance.

5 Conclusions and Future Work

We introduced a novel framework for applying state and action sequence constraints to MDPs using formal languages, supporting both soft and hard constraints. We also introduced a variety of methods to augment the MDP state space so reinforcement learners can learn to maximize rewards while respecting the constraints. Our empirical results on multiple constraints in multiple environments show that it is possible to train a policy to significantly reduce the rate of constraint violations when no constraint enforcement (ce) is applied. Also, in many of our results, it is possible to specify constraints to improve reward by, during training, constraining the agents away from action sequences that are known to be unhelpful.

References

[1] Pieter Abbeel, Adam Coates, and Andrew Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *I. J. Robotics Res.*, 29(13):1608–1639, 2010.

[2] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 22–31. PMLR, 2017.

[3] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Proceedings of AAAI 18*, pages 2669–2678, 2018.

[4] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

[5] N. Aréchiga and B. Krogh. Using verified control envelopes for safe controller design. In *2014 American Control Conference*, pages 2918–2923, 2014.

[6] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model-checking algorithms for continuous-time markov chains. *IEEE Transactions on software engineering*, (6):524–541, 2003.

[7] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. *arXiv preprint arXiv:1805.08328*, 2018.

[8] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, jun 2013.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[10] Alberto Camacho, Oscar Chen, Scott Sanner, and Sheila A McIlraith. Decision-making with non-markovian rewards: From ltl to automata-based reward shaping. In *Proceedings of the Multi-disciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, pages 279–283, 2017.

[11] Alberto Camacho, Oscar Chen, Scott Sanner, and Sheila A. McIlraith. Non-markovian rewards expressed in LTL: guiding search via reward shaping. In *Proceedings of the Tenth International Symposium on Combinatorial Search, SOCS 2017, 16-17 June 2017, Pittsburgh, Pennsylvania, USA.*, pages 159–160, 2017.

[12] Dotan Di Castro, Aviv Tamar, and Shie Mannor. Policy gradients with variance related risk criteria. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*, 2012.

[13] Feng Chen and Grigore Roşu. Mop: An efficient and generic runtime verification framework. In *Proceedings of the 22Nd Annual ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications*, OOPSLA '07, 2007.

[14] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, 2001.

[15] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

[16] Kurt Driessens and Saso Dzeroski. Integrating guidance into relational reinforcement learning. *Machine Learning*, 57(3):271–304, 2004.

[17] Fernando Fernández, Javier García, and Manuela M. Veloso. Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems*, 58(7):866–871, 2010.

[18] J. Garcia and F. Fernandez. Safe exploration of state and action spaces in reinforcement learning. *J. Artif. Intell. Res.*, 45:515–564, 2012.

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

[20] Chris Gaskett. Reinforcement learning under circumstances beyond its control. 2003.

[21] Clement Gehring and Doina Precup. Smart exploration in reinforcement learning using absolute temporal difference errors. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13, Saint Paul, MN, USA, May 6-10, 2013*, pages 1037–1044, 2013.

[22] Peter Geibel and Fritz Wysotzki. Risk-sensitive reinforcement learning applied to control under constraints. *J. Artif. Intell. Res.*, 24:81–108, 2005.

[23] Alborz Geramifard, Josh Redding, and Jonathan P. How. Intelligent cooperative control architecture: A framework for performance improvement using safe learning. *Journal of Intelligent and Robotic Systems*, 72(1):83–103, 2013.

[24] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 855–864, 2016.

[25] Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-constrained neural fitted q-iteration. *CoRR*, abs/1809.07823, 2018.

[26] Michael Huth and Marta Z. Kwiatkowska. Quantitative analysis and model checking. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 - July 2, 1997*, pages 111–122, 1997.

[27] Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. Teaching multiple tasks to an RL agent using LTL. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pages 452–461, 2018.

[28] Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 2112–2121, 2018.

[29] Nils Jansen, Bettina Könighofer, Sebastian Junges, and Roderick Bloem. Shielded decision-making in mdps. *arXiv preprint arXiv:1807.06096*, 2018.

[30] Yoshinobu Kadota, Masami Kurano, and Masami Yasuda. Discounted markov decision processes with utility constraints. *Computers & Mathematics with Applications*, 51(2):279–284, 2006.

[31] Gregory Kuhlmann, Peter Stone, Raymond Mooney, and Jude Shavlik. Guiding a reinforcement learner with natural language advice: Initial results in robocup soccer. In *The AAAI-2004 workshop on supervisory control of learning and adaptive systems*. San Jose, CA, 2004.

[32] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism: Probabilistic symbolic model checker. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 200–204. Springer, 2002.

[33] Edith LM Law, Melanie Coggan, Doina Precup, and Bohdana Ratitch. Risk-directed exploration in reinforcement learning. *Planning and Learning in A Priori Unknown or Dynamic Domains*, page 97, 2005.

[34] Xiao Li, Yao Ma, and Calin Belta. Automata guided reinforcement learning with demonstrations. *CoRR*, abs/1809.06305, 2018.

[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 2017, Vancouver, BC, Canada, September 24-28, 2017*, pages 3834–3839, 2017.

[36] Michael L. Littman, Ufuk Topcu, Jie Fu, Charles Lee Isbell Jr., Min Wen, and James MacGlashan. Environment-independent task specifications via GLTL. *CoRR*, abs/1704.04341, 2017.

[37] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[38] Teodor Mihai Moldovan and Pieter Abbeel. Safe exploration in markov decision processes. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*, 2012.

[39] Seyed Sajad Mousavi, Behzad Ghazanfari, Nasser Mozayani, and Mohammad Reza Jahed-Motlagh. Automatic abstraction controller in reinforcement learning agent via automata. *Appl. Soft Comput.*, 25:118–128, 2014.

[40] 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 1999), Bled, Slovenia, June 27 - 30, 1999*, pages 278–287, 1999.

[41] Arnab Nilim and Laurent El Ghaoui. Robust control of markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005.

[42] Michael T Rosenstein and Andrew G Barto. Supervised learning combined with an actor-critic architecture. *Department of Computer Science, University of Massachusetts, Tech. Rep*, pages 02–41, 2002.

[43] Makoto Sato, Hajime Kimura, and Shibenobu Kobayashi. Td algorithm for the variance of return and mean-variance reinforcement learning. *Transactions of the Japanese Society for Artificial Intelligence*, 16(3):353–362, 2001.

[44] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[45] Aviv Tamar, Huan Xu, and Shie Mannor. Scaling up robust mdps by reinforcement learning. *CoRR*, abs/1306.6189, 2013.

[46] Jie Tang, Arjun Singh, Nimbus Goehausen, and Pieter Abbeel. Parameterized maneuver learning for autonomous helicopter flight. In *IEEE International Conference on Robotics and Automation, ICRA 2010, Anchorage, Alaska, USA, 3-7 May 2010*, pages 1142–1148, 2010.

[47] Andrea Lockerd Thomaz and Cynthia Breazeal. Teachable robots: Understanding human teaching behavior to build more effective robot learners. *Artif. Intell.*, 172(6-7):716–737, 2008.

[48] Lisa Torrey and Matthew E Taylor. Help an agent out: Student/teacher learning in sequential decision tasks. In *Proceedings of the Adaptive and Learning Agents workshop (at AAMAS-12)*, 2012.

[49] Min Wen, Ivan Papusha, and Ufuk Topcu. Learning from demonstrations with high-level side information. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 3055–3061, 2017.

[50] Eric Wiewiora, Garrison W Cottrell, and Charles Elkan. Principled methods for advising reinforcement learning agents. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 792–799, 2003.

Table 4: Per-episode rewards for Atari environments and soft constraints with reward shaping $r_c = -1$

Environments	Constraints	Baseline		Reward Shaping (Violation Penalty $= -1$)							
				Shaping Only		State Augmentation					
						action history		dfa one-hot		dfa embedding	
		rewards	violations	rewards	violations	rewards	violations	rewards	violations	rewards	violations
Breakout	Actuation	27.06 ± 12.38	60.82 ± 336.80	**32.12 ± 13.05**	1.72 ± 22.72	27.68 ± 13.30	3.01 ± 29.20	24.41 ± 10.82	**0.33 ± 0.61**	24.35 ± 12.13	3.40 ± 31.73
	Dithering	27.97 ± 13.09	68.37 ± 532.01	**29.87 ± 13.13**	0.43 ± 0.72	23.45 ± 10.56	0.70 ± 1.40	28.00 ± 13.26	**0.16 ± 0.37**	20.76 ± 10.48	0.18 ± 0.40
Space Invader	Actuation	248.76 ± 124.82	99.36 ± 53.49	230.64 ± 104.55	14.93 ± 11.28	282.49 ± 134.01	6.29 ± 5.46	249.28 ± 152.46	**0.93 ± 1.16**	**323.30 ± 168.01**	2.52 ± 2.23
	Dithering	274.73 ± 175.72	3.39 ± 3.27	**306.62 ± 175.38**	2.71 ± 2.08	249.41 ± 121.10	0.77 ± 0.93	223.58 ± 123.78	**0.34 ± 0.60**	227.27 ± 122.69	0.36 ± 0.62
Seaquest	Actuation	940.43 ± 691.40	57.69 ± 37.79	1230.53 ± 866.39	32.65 ± 19.68	1161.07 ± 677.36	14.03 ± 9.52	970.77 ± 632.48	**0.99 ± 1.48**	**1548.76 ± 797.89**	24.42 ± 16.27
	2d-dithering	1307.50 ± 936.24	6.13 ± 5.70	**1382.31 ± 734.02**	5.53 ± 3.52	1055.24 ± 543.70	1.36 ± 1.29	1290.80 ± 634.89	**0.94 ± 1.11**	1081.62 ± 547.72	4.25 ± 5.17

Table 5: Per-episode rewards for Atari environments and soft constraints with reward shaping $r_c = -10$

Environments	Constraints	Baseline		Reward Shaping (Violation Penalty $= -10$)							
				Shaping Only		State Augmentation					
						action history		dfa one-hot		dfa embedding	
		rewards	violations	rewards	violations	rewards	violations	rewards	violations	rewards	violations
Breakout	Actuation	**27.06 ± 12.38**	60.82 ± 336.80	24.23 ± 12.88	4.85 ± 36.28	17.86 ± 9.11	4.55 ± 49.06	21.27 ± 9.59	**0.23 ± 0.48**	19.36 ± 8.98	2.97 ± 106.78
	Dithering	**27.97 ± 13.09**	68.37 ± 532.01	25.54 ± 12.46	0.27 ± 0.50	26.99 ± 12.58	0.20 ± 0.43	26.45 ± 11.87	**0.15 ± 0.37**	27.45 ± 12.80	0.18 ± 0.41
Space Invader	Actuation	248.76 ± 124.82	99.36 ± 53.49	266.62 ± 105.53	16.81 ± 15.34	234.50 ± 122.84	6.18 ± 5.67	248.44 ± 123.18	**0.64 ± 0.91**	**308.48 ± 144.38**	1.08 ± 1.28
	Dithering	274.73 ± 175.72	3.39 ± 3.27	246.56 ± 144.70	2.18 ± 1.89	271.08 ± 182.96	0.76 ± 0.99	251.66 ± 137.83	**0.33 ± 0.59**	**309.72 ± 168.20**	**0.33 ± 0.59**
Seaquest	Actuation	940.43 ± 691.40	57.69 ± 37.79	1389.10 ± 845.49	14.56 ± 11.80	1589.61 ± 685.61	4.67 ± 3.71	1474.23 ± 680.18	**0.75 ± 1.24**	**1643.71 ± 697.86**	12.03 ± 9.95
	2d-dithering	1307.50 ± 936.24	6.13 ± 5.70	632.90 ± 803.84	3.45 ± 3.83	**1456.64 ± 689.19**	1.29 ± 1.30	1310.88 ± 874.82	**0.93 ± 1.10**	1032.56 ± 808.08	2.06 ± 1.90

Table 6: Per-episode rewards for Atari environments and soft constraints with reward shaping $r_c = -100$

Environments	Constraints	Baseline		Reward Shaping (Violation Penalty $= -100$)							
				Shaping Only		State Augmentation					
						action history		dfa one-hot		dfa embedding	
		rewards	violations	rewards	violations	rewards	violations	rewards	violations	rewards	violations
Breakout	Actuation	**27.06 ± 12.38**	60.82 ± 336.80	9.66 ± 7.45	50.13 ± 138.97	13.92 ± 6.94	0.71 ± 1.61	15.52 ± 9.14	**0.23 ± 0.48**	15.03 ± 9.00	3.85 ± 123.79
	Dithering	**27.97 ± 13.09**	68.37 ± 532.01	16.80 ± 6.84	0.23 ± 0.46	13.22 ± 8.35	0.24 ± 0.51	16.61 ± 6.54	**0.15 ± 0.37**	17.49 ± 6.94	0.15 ± 0.37
Space Invader	Actuation	248.76 ± 124.82	99.36 ± 53.49	237.41 ± 100.43	11.61 ± 16.44	245.16 ± 114.04	3.39 ± 4.06	**250.99 ± 150.48**	**0.64 ± 0.91**	216.27 ± 132.40	0.98 ± 1.18
	Dithering	**274.73 ± 175.72**	3.39 ± 3.27	222.74 ± 128.29	2.56 ± 1.90	274.59 ± 130.07	0.58 ± 0.78	244.99 ± 149.52	**0.33 ± 0.59**	206.50 ± 143.11	0.33 ± 0.59
Seaquest	Actuation	940.43 ± 691.40	57.69 ± 37.79	1085.26 ± 579.68	29.64 ± 26.70	1211.84 ± 778.90	2.98 ± 3.54	**1367.67 ± 795.80**	**0.11 ± 0.40**	1340.12 ± 894.52	18.59 ± 29.03
	2d-dithering	1307.50 ± 936.24	6.13 ± 5.70	**1621.43 ± 840.11**	6.68 ± 4.99	1118.29 ± 500.60	1.51 ± 1.41	1234.05 ± 701.23	**0.93 ± 1.10**	1158.97 ± 1138.63	2.75 ± 3.75

Table 7: Per-episode rewards for Atari environments and soft constraints with reward shaping $r_c = -1000$

Environments	Constraints	Baseline		Reward Shaping (Violation Penalty $= -1000$)							
				Shaping Only		State Augmentation					
						action history		dfa one-hot		dfa embedding	
		rewards	violations	rewards	violations	rewards	violations	rewards	violations	rewards	violations
Breakout	Actuation	**27.06 ± 12.38**	60.82 ± 336.80	2.35 ± 1.95	38.15 ± 122.20	3.17 ± 2.40	12.21 ± 68.50	3.04 ± 2.27	**0.23 ± 0.48**	3.38 ± 3.64	9.73 ± 53.69
	Dithering	**27.97 ± 13.09**	68.37 ± 532.01	5.98 ± 4.03	0.17 ± 0.40	7.00 ± 4.03	0.19 ± 1.74	5.42 ± 4.55	**0.15 ± 0.37**	4.65 ± 4.27	0.16 ± 0.37
Space Invader	Actuation	248.76 ± 124.82	99.36 ± 53.49	244.55 ± 141.62	16.68 ± 16.87	**273.06 ± 128.07**	5.73 ± 4.22	234.59 ± 139.82	**0.64 ± 0.91**	169.98 ± 121.59	0.97 ± 1.25
	Dithering	**274.73 ± 175.72**	3.39 ± 3.27	247.57 ± 198.57	2.66 ± 2.14	251.67 ± 132.38	1.20 ± 1.67	258.98 ± 140.54	**0.33 ± 0.59**	199.29 ± 130.66	0.35 ± 0.61
Seaquest	Actuation	940.43 ± 691.40	57.69 ± 37.79	1011.86 ± 717.78	25.04 ± 21.57	**1343.50 ± 636.67**	6.90 ± 6.17	1107.85 ± 620.55	**0.10 ± 0.33**	1299.00 ± 692.49	17.16 ± 14.20
	2d-dithering	1307.50 ± 936.24	6.13 ± 5.70	956.59 ± 807.72	6.54 ± 5.71	982.58 ± 689.03	1.79 ± 1.72	**1353.97 ± 724.53**	**0.93 ± 1.10**	1163.53 ± 820.72	2.15 ± 2.21