Guiding Safe Reinforcement Learning Policies Using Structured Language Constraints

Bharat Prakash[1], **Nicholas Waytowich**[2], **Ashwinkumar Ganesan**[1], **Tim Oates** [1], **Tinoosh Mohsenin**[1]
[1]University of Maryland, Baltimore County (UMBC), [2] US Army Research Laboratory,

Abstract

Reinforcement learning (RL) has shown success in solving complex sequential decision making tasks when a well defined reward function is available. For agents acting in the real world, these reward functions need to be designed very carefully to make sure the agents act in a safe manner. This is especially true when these agents need to interact with humans and perform tasks in such settings. However, hand-crafting such a reward function often requires specialized expertise and quickly becomes difficult to scale with task-complexity. This leads to the long-standing problem in reinforcement learning known as *reward sparsity* where sparse or poorly specified reward functions slow down the learning process and lead to sub-optimal policies and unsafe behaviors. To make matters worse, reward functions often need to be adjusted or re-specified for each task the RL agent must learn. On the other-hand, it's relatively easy for people to specify using language what you should or shouldn't do in order to do a task safely. Inspired by this, we propose a framework to train RL agents conditioned on constraints that are in the form of structured language, thus reducing effort to design and integrate specialized rewards into the environment. In our experiments, we show that this method can be used to ground the language to behaviors and enable the agent to solve tasks while following the constraints. We also show how the agent can transfer these skills to other tasks.

Introduction

Reinforcement learning (RL) has shown great success in learning policies for complex tasks where the behaviors cannot easily be designed *a-priori*, such as training an agent to navigate a 3D environment or dextrous object manipulation. Using RL, the agent learns policies that optimize future rewards by exploring the environment and getting feedback from its interactions (Sutton, Barto, and others 1998). As the environments grow in complexity, the policies that govern agent behavior grow in complexity too. Not only do they encompass the physics of how agents can operate in the environment, but they also define how to do so safely. As embodied AI agents (i.e. robots) become increasingly more integrated in our society, it is vital that the behaviors that these AI systems learn are safe. *Reward shaping* is a method for shaping agent behavior by modifying the reward function that the RL agent attempts to maximize. However, it's unclear how to effectively train safe behaviors using reward shaping alone as an agent must be able to predict harm that might occur before it takes a dangerous action, not after. Additionally, since these AI systems will operate alongside humans, it is important to allow for non-technically trained people to be able to easily specify AI behavior. Current reward shaping approaches require specialized knowledge to engineer rules by hand for every task and environment the AI needs to solve.

Conversely, humans receive instructions and guidance through natural language while internalizing the knowledge gained from interacting with the external world. We have the capability to adapt our behavior, find alternate routes and use the same warnings across scenarios (with similar conditions) even when the warnings do not exist. For example, we know how to navigate sidewalks and operate doors in buildings. However, when we see signs such as "sidewalk closed ahead" or "don't open door" we can use a different door or find a different sidewalk and still reach our destination. Also, we can find a different sidewalk while walking a new street if we see that there is construction work on the street even though a sign is not present.

In this work, we propose an architecture to train agents who can ground structured language constraints to behavior and learn a policy to avoid going into unsafe states. In order to do this, we first collect a dataset of trajectory segments and structured language constraints. These are then labeled by a human based on whether or not violations occurred. This data set is used to train what we call a "constraint checker", which is an embedding model used to provide an auxiliary reward signal during training of the RL agent. Using a neural network architecture, we train the constraint checker, which maps complex behaviors to constraints using language, to detect unsafe behaviors which might be difficult to convey otherwise using hand written rules. This method can then be used to train RL agents to learn safe behaviors.

In our proposed architecture, the constraint checker is used while training the RL agent with different structured language constraints and random task initializations. During test time, we show that the agent performs the task while following the constraints provided each time. Additionally, we test the generalization of our method and show that the same constraint checker can be used in different environments and

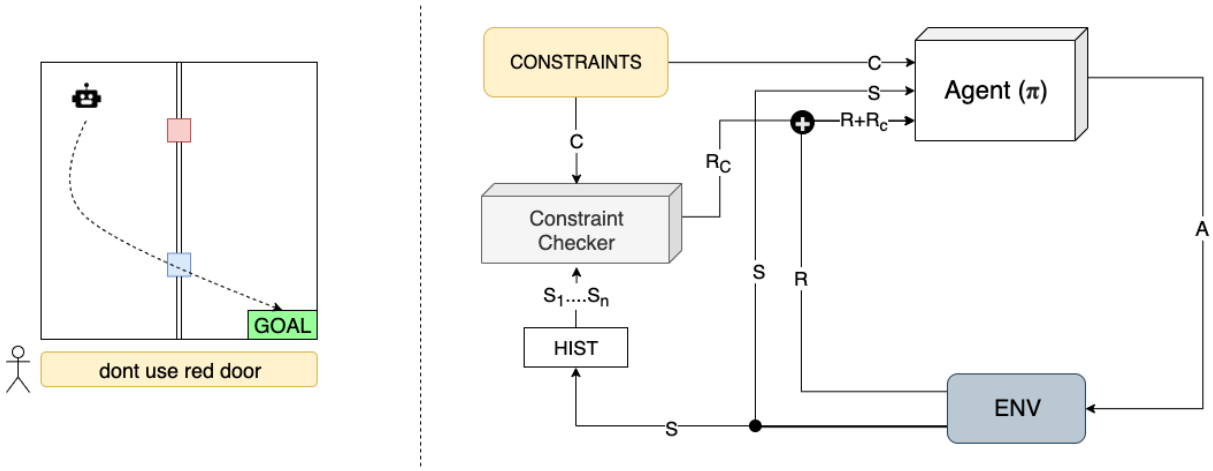


Figure 1: Example scenario and system architecture. Consider a scenario where the agent needs to navigate rooms and reach a destination. As shown on the left, there might be multiple paths to reach the destination. The agent can either use the red door or the blue door. Now, the agent also receives a constraint in the form of language, "don't use red door". The agent will have to use the blue door instead and reach the goal. Our system receives language constraints and grounds them to a behavior. To do this, we use the constraint checker module during training which can interpret agent behavior along with the language constraint and shape environment rewards to learn safe behaviors. HIST stores the current and $n-1$ prior states. The constraint checker receives the state sequence (from HIST) and the structured language constraint and outputs a reward R_c. The agent receives this additional reward signal from the constraint checker along with the state and language constraint.

train the agents to follow similar constraints. We test this on the Mini-Grid environment where we generate random constraints in English. The experiments were done on 3 scenarios with different difficulty levels in terms of tasks and language constraints.

Related Work

In most reinforcement learning and sequential decision making problems, tasks are specified either using reward functions (Sutton, Barto, and others 1998) or demonstrations (Argall et al. 2009). Some other ways to specify tasks use human feedback in the form of preferences as shown in (Christiano et al. 2017) and (Gandhi et al. 2019). On the other hand, using language instructions is a very appealing way to specify tasks and goals. It is very natural and easy for humans to use language to convey goals and intent. You don't need to be an expert in actually performing the task and you don't need to manually engineer or design reward functions. Unlike learning from preferences, natural language specifications can encode more information about the task and the way to perform the task.

Language can be used to specify plans, goals, and high level requirements to each other (Gopnik and Meltzoff 1987). We humans can learn to do tasks in new environments not only from demonstrations, but also from information encoded using language (Tsividis et al. 2017). Traditional Reinforcement Learning and Imitation Learning approaches don't really attempt to ground language and environment features. It is important to develop intelligent agents with the same capabilities.

In most current research, language is used in RL in two main ways, language conditioned RL and language-assisted RL (MacMahon, Stankiewicz, and Kuipers 2006) (Hermann et al. 2017). Methods developed for language conditional tasks are relevant for language-assisted RL as they both deal with the problem of grounding natural language sentences in the context of RL (Goyal, Niekum, and Mooney 2019) (Bahdanau et al. 2018). Instruction following agents are presented with tasks defined by high-level (sequences of) instructions based on language. Most techniques focus on instructions that are represented by (at least somewhat natural) language, and may take the form of formal specifications of appropriate actions, of goal states (or goals in general), or of desired policies. Another use of instructions is to induce a reward function for RL agents or planners to optimize. We will be exploring more of this type of a problem.

There has been some work in learning mappings between language and actions or behaviors. Branavan et al. (2009) shows a way to learn mapping between language instructions and action in reinforcement learning. They show this on simple game tutorials and troubleshooting environments. Chen and Mooney (2011) present a system that learns a semantic parser for interpreting navigation instructions by simply observing the actions of human followers and recent work has shown the utility of natural language narrations for guiding RL policies to learn complex tasks such as StarCraft 2 (Waytowich et al. 2019b; 2019a).

Ensuring safety in reinforcement learning algorithms is also an important area of research. There has been a lot of effort in building safe agents which avoid going into dangerous states and during training as well as inference. This is especially true when the agent works and interacts closely with humans. Saunders et al. (2017) show ways to use a human feedback to build a model which blocks unsafe actions

and try to learn tasks safely. Prakash et al. (2019b) extend this work by combining model based and model free reinforcement learning to improve safety. Amodei et al. (2016) and Leike et al. (2017) outline and motivate safe artificial intelligence research and also provide some test environments.

Background

Reinforcement learning provides a framework for decision-making and control where an agent tries to maximize long-term rewards by interacting with a complex environment. This can be applied to various tasks such as autonomous vehicles and robotics. Unlike other machine learning paradigms, there is no supervisor and we only provide a high level objective function. The feedback might be delayed, the data is not independent identically distributed (i.i.d) and the agent's actions affect the subsequent data it receives. A policy is the agent's behavior function or the mapping from a state to action. A Value function is used to evaluate how good a state is; it is the prediction of future rewards. The dynamics model is the model of the environment or world, it is a function which predicts the next state and reward, given the current state and action. An RL agent can consist of one or more of these components.

Usually, reinforcement learning is formalized using Markov Decision Processes (MDP). An MDP is defined as a tuple (S, A, P, R, γ), where S is the state space, A is the action space or the set of actions available to an agent, P is the unknown transition function, R is the reward function and γ ϵ (0, 1) is the discount factor. The RL agent interacts with the environment by acting according to a policy π which is a mapping from states to actions, or a probability distribution over actions. The goal at each step is to maximize the discounted sum of future rewards, $\sum_{t'=t}^{\infty} \gamma^{t'-t} R_{t'}$, and the quality of the policy at time t is measured by the value function $E_{\pi}(\sum_{t}^{\infty} \gamma^{t} R_{t+1} | s_0 = s)$, with starting state s.

We follow a similar structure described above. However, the agent also receives a constraint or a warning in the form of a structured language sentence C. The agent needs to learn a policy $\pi(a|s, C)$ which can perform the task while satisfying these constraints. We define a trajectory segment, T as a sequence of states which might represent the agents behavior. For example, if the trajectory segment is of length 5, T_i is the trajectory segment at time step $i (i > 5)$ which is a sequence of 5 states $S_{i-5} ... S_i$.

In the next sections, we will explain the architecture in more detail along with the experiments and results.

System Architecture

The proposed architecture has two main components. The constraint checker module and the RL policy module. The constraint checker module is used to determine whether a given constraint (i.e. "do not use the red door") was violated or not and is used to augment the environment's reward function or can be used as a proxy to the environment reward. It takes a trajectory segment T_i (or a state sequence) and a structured language constraint C and outputs a binary label indicating violations. This output is used by the RL agent as an auxiliary reward signal to learn safe policies.

Data Collection

The data to train the constraint checker needs to be collected using human annotators. In order to train the constraint checker, we need samples of trajectory segments and language constraints as input and binary label as the output. This data can be collected in various ways depending on the environment. In simulated environments, the agent and goal positions can be randomized and the constraints can be generated from a fixed vocabulary and grammar. The human annotator can be presented with these pairs and be asked to label whether or not the agent violated a constraint as well as which constraint was violated. Likewise, the human can be presented with just the trajectories and asked to come up with possible structured constraints. Alternatively, it might be easier to use a human to demonstrate behaviors themselves and then label them with structured constraints.

In this paper, we automated the process of annotating violations since we had access to the simulated environment back end. A partially trained agent was rolled out in the environment and random structured constraints were generated at each episode. The data was collected to make sure we have enough samples of all types are language constraints and agent behaviors. In this dataset, each sample consists of a sequence of states, S_1 to S_n which corresponds to the agent's behavior and a binary label which denotes whether or not a violation occurred given the random language constraint. Additionally, instead of labelling behaviors as violations after they happen, we set up our annotation routine to label something as a violation just before it was about to happen. We hypothesize that training the constraint checker using this approach can potentially reduce the amount violations in the training phase. We validate this in the Results section.

Constraint Checker

The constraint checker module interprets the language constraint C and a trajectory segment T and outputs whether or not a violation occurred. This label can then be used to augment the environment reward to train agents which can avoid violations and act safely in the environment.

Inside the constraint checker are two sub-modules: the language module and the trajectory module as shown in 2. The language module accepts the structured language constraint and outputs a sentence embedding. The trajectory module takes in as input a trajectory (sequence of states) and generates a trajectory embedding.

The language module consists of an embedding layer of size 32 and learns the word embeddings from the vocabulary of our constraints (see Table 1). These are then passed through a GRU layer of size 128 to get the sentence embedding. The trajectory segment T is a sequence of states as we described earlier. Each state (a 2D image in our case) is processed by 2D Convolution layers to get a sequence of state embeddings of size 64. This is then is passed through a 1D Convolution to generate a trajectory embedding. The sentence embedding and the trajectory embedding are concatenated and then passed through an MLP to give us the final label representing violations. This module is trained in a supervised fashion from the data collected earlier.

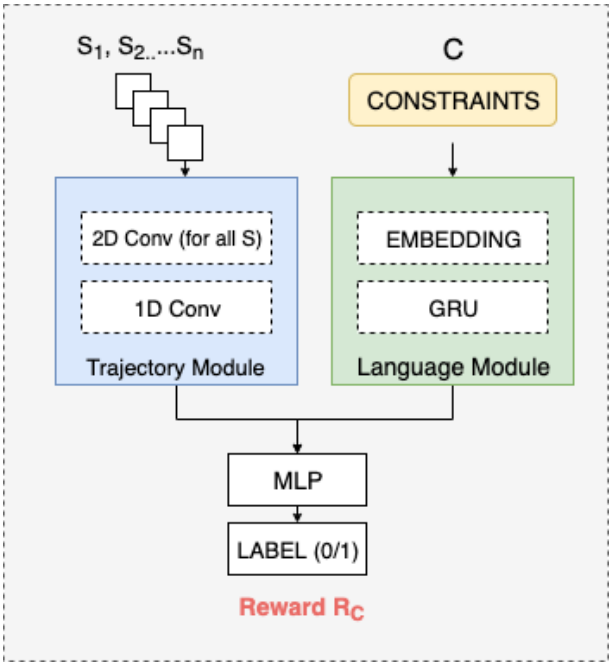


Figure 2: Architecture of the Constraint Checker . It accepts a trajectory segment and a language constraint and outputs a binary label which denotes violations. This can can be used to shape the rewards which the agent receives. The trajectory segment is a sequence of states the agent observes which are processed by 2d convolution layer which gives us a array of state embeddings. This is then processed by a 1d convolution to give us a trajectory embedding. The textual constraint is processed by an embedding layer followed by a Gated Recurrent Unit (GRU) layer to give us a sentence embedding. Both these embeddings are concatenated and processed by a Multi-Layer Perceptron (MLP) to give us the final output label.

Reinforcement Learning Agent

The architecture of our entire model is show in Figure 1. The RL agent receives a state S and the language constraint C and outputs a probability distribution over the actions. It processes the state S using 2D convolution layers followed by a dense layer. And the constraint C is processed by a module similar so the language module shown in the constraint checker architecture. The reward that the RL agent receives is a function of the default environment reward R and the output of the constraint checker R_c. R_c depends on whether or not the constraint checker thinks the agent behavior is bad given the constraint. The RL policy is trained using Proximal policy optimization(Schulman et al. 2017).

Experiments and Results

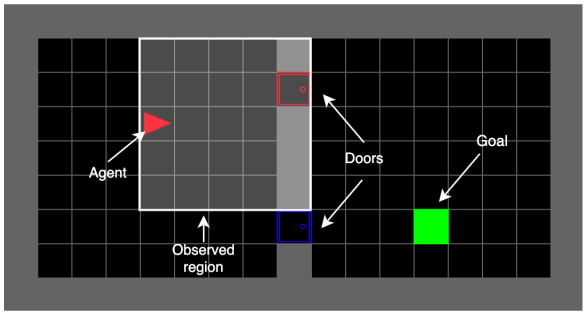
In this section, we will explain the experimental setup as well as the results. Our experiments are performed on the MiniGrid Environment by Chevalier-Boisvert, Willems, and Pal (2018) which is a partially observable grid world environment. The main goal of our experiments is to understand how well our model can learn to interpret language constraints and act safely in the environment. We also evaluate how the constraint checker can be reused in different environments and tasks.

Environmental Setup

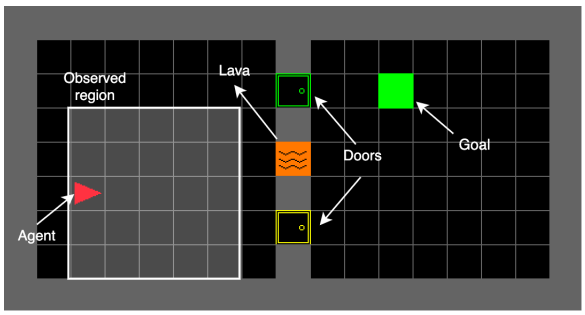
The environment can have multiple rooms with doors, walls and goal objects. The doors can have multiple colors and the agent and goal objects are spawned at random locations. The action space is discrete which allows movement in all 4 directions, opening and closing doors and picking up and dropping objects. The environment is partially observable, the agent can only see an ego-centric 5×5 view in front of the agent. Also, the agent cannot see through walls and closed doors. We designed multiple scenarios in this environment with increasing difficulty levels both in terms of the tasks and the language constraints. We compare the performance of our model with a baseline where we shape the environment rewards directly by giving negative rewards for violations.

The 2 Rooms scenario is shown in the Fig 3 (a). It consists of 2 rooms separated by a wall and 2 doors. In each episode, the agent and the goal are spawned at random locations and the door colors are randomly initiated. Also there is a random language constraint generated which follows a fixed grammar as shown in the examples in Table 1. The task is to reach the green goal object using the minimum number of steps while also going through the correct door (which is specified using the language constraints).

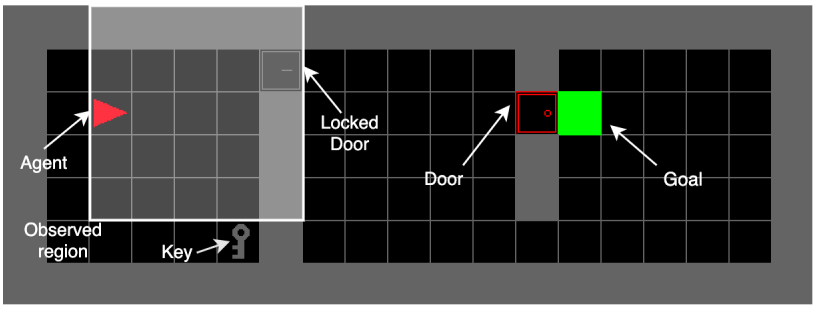
The 2 Rooms with lava scenario is shown in Fig 3 (b). This is similar to the 2 Rooms environment but it has an additional cell called the Lava. It is the orange cell seen in



(a) 2 Rooms: There are 2 rooms separated by a wall and 2 doors. The agent in red needs to reach the green goal location by using one of the doors. And the agent can only see a 5×5 area in front of it.



(b) 2 Rooms-Lava: There are 2 rooms separated by a wall and 2 doors. There is also a lava cell which can behave in different ways. The agent in red needs to reach the green goal location by using one of the doors. And the agent can only see a 5×5 area in front of it. Also, the agent may or may not be allowed to use the lava cell depending on the language constraint.



(c) 3 Rooms-Key: There are 3 rooms separated by walls and 2 doors. The first room has a locked door and needs a key to unlock it. The agent needs to pick up the key before opening the first door. Then, the agent can either use the door or take a longer way to reach the goal state. Again, this depends on the language constraint.

Figure 3: Environments used in the experiments

the figure and it can behave in one of two ways depending on the language constraint. Example language constraints are shown in Table 1. For constraints of type 1-3, the lava acts as a teleportation cell, the agent entering the lava will be instantly moved right next to the agent. If the constraint is of type 4-5, the agent entering the lava will die and result in 0 reward.

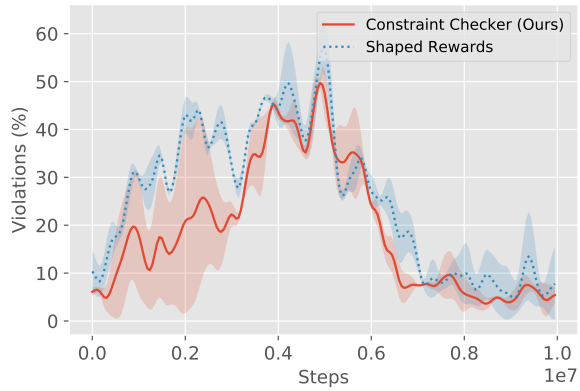
The 3 rooms with key scenario is shown in Fig 3 (c) where the task is to again reach the goal state but the first room is locked. The agent needs to collect the key first and then unlock the door. The goal cell is always in the upper right corners of the environment. If there is no restriction, taking the second door is always the shortest path. The constraints here are similar to the ones in 2 rooms scenario. Depending on the constraint, using the door might be a violation and the agent might have to use a longer route from the bottom of the second wall.

#	Constraints	Environments
1	do not use the red door	2Rooms, Lava, Key
2	do not go through the blue door	2Rooms, Lava, Key
3	no yellow door	Lava
4	do not go through the red door and stay away from lava	Lava
5	avoid blue door and no lava	Lava

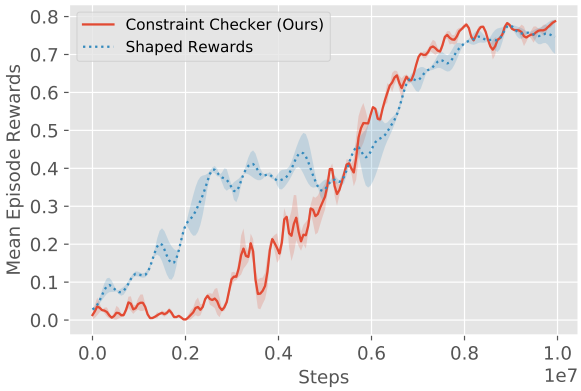
Table 1: Example constraints used in the 3 environments used in our experiments

Experimental Conditions

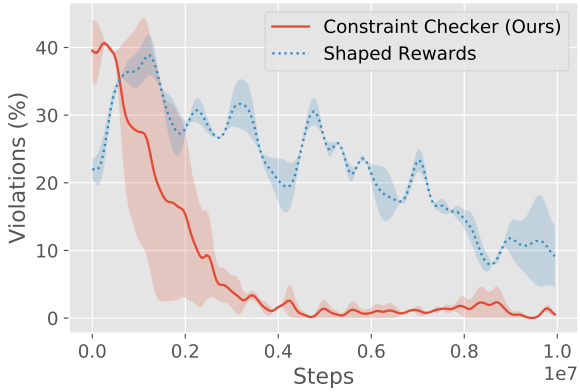
We compare our method with a baseline where we train the agent with shaped environment rewards. This means that the reward function is changed in the MiniGrid framework to output negative rewards whenever a language constraint is violated. This is usually called as reward shaping. In order to do this we need access to the environment reward function (the MiniGrid framework in this case) which is not always feasible in most real-world tasks. The baseline method uses the same RL policy architecture as ours (minus the constraint checker) and is also trained using Proximal policy optimization (PPO). In contrast, our method does not change the default reward function in anyway and thus does not



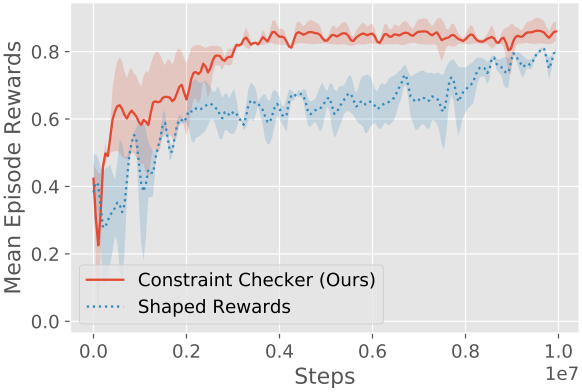
(a) Violations: 2 Rooms



(b) Episode Reward: 2 Rooms



(c) Violations: 2 Rooms-Lava



(d) Episode Reward: 2 Rooms-Lava

Figure 4: Violations % and **mean episode rewards** for the **2 Rooms** (top row) and **2 Rooms-Lava** (bottom row) environments. The constraint Checker is able to optimize agent behavior to reduce violations faster than the shaped rewards baseline. Additionally, our model is able to achieve higher episode rewards on average at the end of training than the shaped rewards method.

rely on the requirement of having access or the expertise to change the reward function. Instead, the constraint checker detects violations and shapes the reward which is given to the agent. In other words, this allows for a more intuitive means of reward-shaping through structured language. The aim of our experiments is to show that our method can perform as good as the baseline in terms of performance while also minimizing the amount of violations our agent incurs.

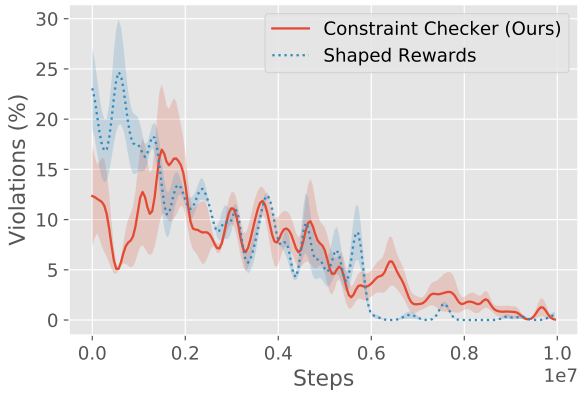
Results

We trained our RL agents using the proposed constraint checker method as well as the baseline method for all three tasks. For the constraint checker, we used 12k samples for the 2 Room environment and 15k samples for the 2 Rooms-Lava environment. Both methods were trained for $1e7$ (10 Millions) steps for all three environments. Then, we recorded the task performance (in terms of baseline reward) and number of constraint violations.

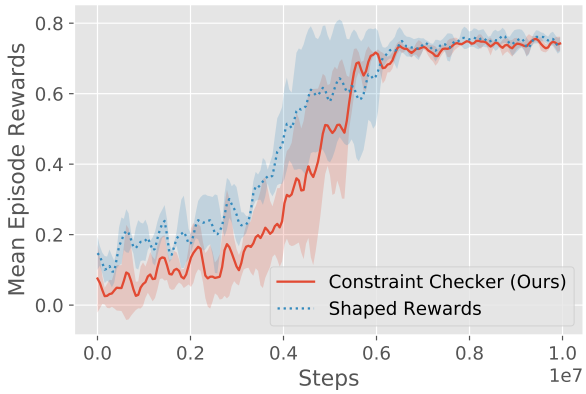
Figures 4 and 5 show the mean violations and rewards of for our constraint checker approach and the reward shaping baseline approach. In order to get these plots, we save the model at regular intervals throughout the training process. We then evaluate these models on 50 random episodes. The violation percentage is the number of times the agent performs a violation given the constraint and the mean episode rewards is over the 50 random episodes. The reward the agent receives is a reward of 1 after reaching the goal state and discounted by the number of steps it takes to reach the goal.

In Figure 4b, we can see that the reward shaping model and our model reach the same performance in terms of rewards at the end of training. This shows that the constraint checker was able to interpret the language constraints and behavior and provide correct rewards that still allow the agent to reach it's goal. We also show that the violation percentage is much less in our method compared to the baseline reward shaped model especially at the beginning of training.

The next experiment was done on a slightly more com-



(a) Violations: 2 Rooms-Lava to 3 Rooms-Key



(b) Reward: 2 Rooms-Lava to 3 Rooms-Key

Figure 5: Generalization Experiment. **Violations %** and **mean episode rewards** for the **3 Rooms-Key** environment. Here the constraint checker from the **2 Rooms** experiment was reused for the 3 rooms-key as a test of generalization. Again, our method is able to optimize agent behavior to reduce violations and achieve reward performance as good as the baseline. This shows how we can re-use the constraint checker to train an agent to perform a different task where we have similar constraints.

plex scenario (2 Rooms-Lava), both in terms of the task and the constraint. Since this scenario has different kinds of constraints and violations, we collected data and trained a new constraint checker. The example of these constraints are shown in Table 1, rows 3-5. The results from the 2 rooms-lava scenario (shown in Figure 4c-d) show that the constraint checker achieves improved performance in terms of episode reward over the baseline method early on. The baseline method is able to eventually catch up and match the performance of the constraint checker. Additionally, we see that using the constraint checker results in drastically lower violations during training. The reason for this, we believe is because the nature of which the constraint checker was trained. The data used to train the constraint checker was collected in such a way that a trajectory segment was labelled as negative, just before the violation was about to happen.

As a test of generalizability, we performed another experiment to show the re-usability of the constraint checker on a new MiniGrid environment. We use the same constraint checker trained in the 2-rooms environment, but apply it to the 3 rooms with key environment where the task and the environment are different but the constraints and violations are the same. The results here show that we were able to re-use the constraint checker which trained for a different scenario (as is evident in Figure 5 with steep training curves that converge quickly to maximum performance as well as to near zero constraint violations). Again, the reward and the violation percentage of our method matches the baseline reward shaped model. This demonstrates that we were able to take our constraint checker, pre-trained on a different task, and achieve high reward performance on a new task with minimal violations.

Discussion

It is really important that agents acting in the real world have the capability to understand instructions and warning signs in the form of natural language. This work presents an architecture which can learn mappings between simple language constraints and agent behavior. This is then used to train agents who can solve tasks without violating these constraints. In order to do this, we collect a dataset of agent behavior and language constraints and train a constraint checker model in a supervised fashion. This is then used to shape rewards and train agents to interpret natural language constraints and act safely. Currently, we test this on small vocabulary of structured sentences, however future work will involve extending this to more complex state spaces and language.

There has been similar work done where language is used to help train reinforcement learning policies. Most of these however, attempt to use language to tackle issues with sparse rewards and the long term credit assignment problem. They use language instructions as a way to guide exploration during training and accelerate learning. In our work, we instead focus on using language to constrain the policy instead of simply guide it. The benefit of this is that our language constraints are used to train agents to avoid certain behaviors and ultimately act safely in environments.

Saunders et al. (2017) and Prakash et al. (2019b) have done work in safe reinforcement learning where they use a method to learn a mapping between state action pairs and safety. This can be useful to avoid going into bad states during exploration. However, since it is only using state action pairs it can be difficult to use these methods to specify safety for more complex behaviors. Our proposed method on the other hand, can ground a structured language sentence in terms of agent behavior and has the potential to achieve broader expressivity in terms of behaviors that should be ei-

ther preferred or avoided. This is a step towards building agents who can interpret language and learn safe behaviors using human feedback.

One limitation of this work is that it requires a way to either generate data automatically to train the constraint checker or a person to collect and annotate the data. In sufficiently complex scenarios, it will not feasible to automate the curating of such a dataset and must be instead be collected manually. If we wanted to extend this approach to more complex environments or real world problems, annotations will have to be done by a human. Crowd-sourcing platforms, such as Amazon's Mechanical Turk (AMT), can be used to get a large and diverse dataset for supervised training of the constraint checker. Another benefit of this type of annotation is that the human can label more complex behaviors which may be too difficult to specify using hand written rules. Future work will incorporate crowd-sourcing techniques, as well as reducing the amount of human annotations needed in the first place.

Acknowledgments

This project was sponsored by the U.S. Army Research Laboratory under Cooperative Agreement Number W911NF-10-2-0022. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. We also thank Sunil Gandhi for useful discussions during the course of this project.

References

Amodei, D.; Olah, C.; Steinhardt, J.; Christiano, P.; Schulman, J.; and Mané, D. 2016. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*.

Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and autonomous systems* 57(5):469–483.

Bahdanau, D.; Hill, F.; Leike, J.; Hughes, E.; Hosseini, A.; Kohli, P.; and Grefenstette, E. 2018. Learning to understand goal specifications by modelling reward. *arXiv preprint arXiv:1806.01946*.

Branavan, S. R.; Chen, H.; Zettlemoyer, L. S.; and Barzilay, R. 2009. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, 82–90. Association for Computational Linguistics.

Chen, D. L., and Mooney, R. J. 2011. Learning to interpret natural language navigation instructions from observations. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*.

Chevalier-Boisvert, M.; Willems, L.; and Pal, S. 2018. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid.

Christiano, P. F.; Leike, J.; Brown, T.; Martic, M.; Legg, S.; and Amodei, D. 2017. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, 4299–4307.

Gandhi, S.; Oates, T.; Mohsenin, T.; and Waytowich, N. R. 2019. Learning behaviors from a single video demonstration using human feedback.

Gopnik, A., and Meltzoff, A. 1987. The development of categorization in the second year and its relation to other cognitive and linguistic developments. *Child development* 1523–1531.

Goyal, P.; Niekum, S.; and Mooney, R. J. 2019. Using natural language for reward shaping in reinforcement learning. *arXiv preprint arXiv:1903.02020*.

Hermann, K. M.; Hill, F.; Green, S.; Wang, F.; Faulkner, R.; Soyer, H.; Szepesvari, D.; Czarnecki, W. M.; Jaderberg, M.; Teplyashin, D.; et al. 2017. Grounded language learning in a simulated 3d world. *arXiv preprint arXiv:1706.06551*.

Hosseini, M.; Horton, M.; Paneliya, H.; Kallakuri, U.; Homayoun, H.; and Mohsenin, T. 2019a. On the complexity reduction of dense layers from $o(n^2)$ to $o(nlogn)$ with cyclic sparsely connected layers. In *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM.

Hosseini, M.; Paneliya, H.; Kallakuri, Uttej Khatwani, M.; and Mohsenin, T. 2019b. Minimizing classification energy of binarized neural network inference for wearable devices. In *2019 20th International Symposium on Quality Electronic Design (ISQED)*. IEEE.

Khatwani, M.; Hairston, W. D.; Waytowich, N.; and Mohsenin, T. 2019. A low complexity automated multi-channel eeg artifact detection using eegnet. In *2019 IEEE EMBS Conference on Neural Engineering*. IEEE.

Leike, J.; Martic, M.; Krakovna, V.; Ortega, P. A.; Everitt, T.; Lefrancq, A.; Deepmind, L. O.; and Deepmind, S. L. 2017. AI Safety Gridworlds. Technical report.

MacMahon, M.; Stankiewicz, B.; and Kuipers, B. 2006. Walk the talk: Connecting language, knowledge, and action in route instructions. *Def* 2(6):4.

Prakash, B.; Horton, M.; Waytowich, N.; Hairston, W. D.; Oates, T.; and Mohsenin, T. 2019a. On the use of deep autoencoders for efficient embedded reinforcement learning. In *ACM Proceedings of the 29th Edition of the Great Lakes Symposium on VLSI (GLSVLSI)*. ACM.

Prakash, B.; Khatwani, M.; Waytowich, N.; and Mohsenin, T. 2019b. Improving safety in reinforcement learning using model-based architectures and human intervention. In *The 32nd International Conference of the Florida Artificial Intelligence Society (FLAIRS-32)*. AAAI.

Saunders, W.; Sastry, G.; Stuhlmueller, A.; and Evans, O. 2017. Trial without Error: Towards Safe Reinforcement Learning via Human Intervention.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sutton, R. S.; Barto, A. G.; et al. 1998. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge.

Tsividis, P. A.; Pouncy, T.; Xu, J. L.; Tenenbaum, J. B.; and Gershman, S. J. 2017. Human learning in atari. In *2017 AAAI Spring Symposium Series*.

Waytowich, N.; Barton, S. L.; Lawhern, V.; and Warnell, G. 2019a. A narration-based reward shaping approach using grounded natural language commands.

Waytowich, N. R.; Barton, S. L.; Lawhern, V.; Stump, E.; and Warnell, G. 2019b. Grounding natural language commands to starcraft II game states for narration-guided reinforcement learning. *CoRR* abs/1906.02671.