# Automata Conditioned Reinforcement Learning with Experience Replay

**Beyazit Yalcinkaya***
University of California, Berkeley
beyazit@berkeley.edu

**Niklas Lauffer***
University of California, Berkeley
nlauffer@berkeley.edu

**Marcell Vazquez-Chanlatte**
Nissan Advanced Technology Center Silicon Valley
Marcell.VazquezChanlatte@nissan-usa.com

**Sanjit A Seshia**
University of California, Berkeley
sseshia@berkeley.edu

## Abstract

We explore the problem of goal-conditioned reinforcement learning (RL) where goals are represented using deterministic finite state automata (DFAs). Due to the sparse and binary nature of automata-based goals, we hypothesize that experience replay can help an RL agent learn more quickly and consistently in this setting. To enable the use of experience replay, we introduce a novel end-to-end neural architecture, including a graph neural network (GNN) to encode the DFA goal before passing it to a feed-forward policy network. Experimental results in a gridworld domain demonstrate the efficacy of the model architecture and highlight the significant role of experience replay in enhancing the learning speed and reducing the variance of RL agents for DFA tasks.

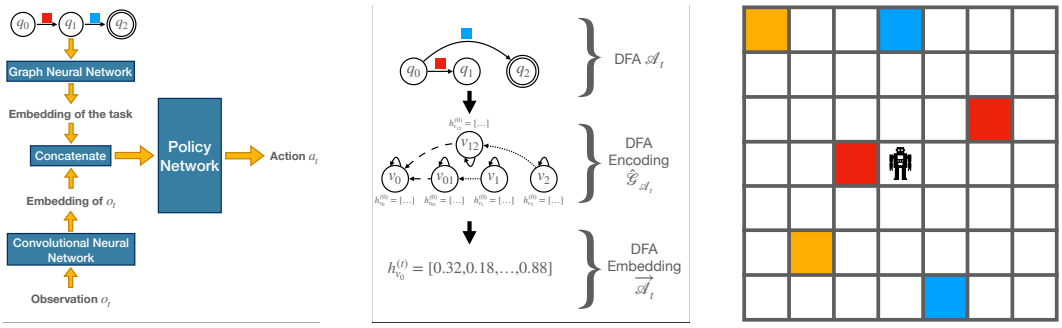
## 1 Introduction

We consider the problem of task-conditioned reinforcement learning (RL) where tasks are encoded as deterministic finite state automata (DFAs). That is, at test time, the agent is given a DFA description of the task and expected to perform the task with no additional training. The focus on DFA-conditioned RL is motivated by three observations. First, many tasks (including non-Markovian ones) can be encoded with finite state automata. For example, many popular formalisms for specifying reinforcement learning (RL) problems encode regular languages and thus can also be represented by DFAs [10, 7]. This class of tasks is of particular interest due to them being closed under Boolean combinations and temporal sequencing while only requiring finite memory to represent. Second, DFAs enable a mechanism to study learning policies conditioned on regular languages without introducing too many (artificial) inductive biases. In particular, DFAs constitute a large and mostly unstructured task representation. This stands in contrast to very structured syntactic task specifications such as finite linear temporal logic [5]. Academically, less inductive bias is valuable for comparing different learning algorithms. Practically, less inductive bias forces the resulting policy to solve the underlying regular language rather than relying on syntactic heuristics. Finally, we are motivated by eventual applications to the inverse problem - learning finite state automata from expert demonstrations [11]. In this setting, the forward problem of knowing the rational behavior of an agent given a DFA is queried hundreds of times on candidate DFAs that are apriori unknown.

*Equal contribution

This work is partially supported by DARPA contracts FA8750-18-C-0101 (AA) and FA8750-23-C-0080 (ANSR) and by Nissan and Toyota under the iCyPhy Center. Niklas Lauffer is supported by an NSF graduate research fellowship.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).




(a) Model architecture (b) A reach-avoid DFA (c) Environment

Figure 1: Visualizations of the proposed framework and the environment used in the experiments.

Like many other goal-conditioned RL problems, automata constitute a *sparse* and *binary* reward. As a consequence, off-the-shelf RL algorithms for automata lack a consistent positive reward signal - a situation made strictly worse by the history-dependent nature of automata. We address this problem using two techniques. First, we leverage the growing literature of Hindsight Experience Replay [2] to enable counterfactual reasoning, i.e., the output behavior is incorrect, but would have been correct for another DFA. Second, we leverage a graph neural network (GNN) to summarise the state of the DFA before passing to a policy network. The intuitive design motivation is that *locally* a DFA looks like a series of reach-avoid problems. Thus, the GNN can identify the active reach-avoid query and condition the policy accordingly. Empirically we observe a significant improvement in the ability of the DFA-conditioned policy network to learn in a grid world domain.

**Related Work.** Prior work has considered leveraging experience replay to policies conditioned on syntactically defined temporal specifications [10]. By treating finite automata, this work studies a less structured concept class where compositional heuristics implicit in Boolean algebra are obscured. Notably, a similar experience replay training strategy has been employed for learning policies for a single reward machine [6] – an automaton with rewards provided at each state. By contrast, we only consider sparse binary feedback for the multi-task setting, i.e., the automaton is provided at runtime.

## 2 Automata Conditioned Reinforcement Learning

The overall architecture of the framework is illustrated in Figure 1a. At step $t$, given an observation $o_t$ and a task represented by a DFA $\mathcal{A}_t$, we compute embeddings $\overrightarrow{o_t}$ and $\overrightarrow{\mathcal{A}_t}$, respectively, and give their concatenation $\overrightarrow{o_t} \cdot \overrightarrow{\mathcal{A}_t}$ to the feed-forward policy network to get a softmax over possible actions. The embedding $\overrightarrow{o_t}$ of the observation $o_t$ is computed by using a *convolutional neural network* (CNN) in the standard way. The embedding $\overrightarrow{\mathcal{A}_t}$ of the DFA $\mathcal{A}_t$ is computed in two steps (cf. Figure 1b) – (i) construct the graph encoding $\hat{\mathcal{G}}_{\mathcal{A}_t}$ of the DFA $\mathcal{A}_t$ and (ii) pass the graph $\hat{\mathcal{G}}_{\mathcal{A}_t}$ through a *graph neural network* (GNN) to get $\overrightarrow{\mathcal{A}_t}$. After taking the action $a_t$ at step $t$, using the next observation $o_{t+1}$, we update the initial state of the DFA $\mathcal{A}_t$ based on the transition taken and minimize it by pruning its unreachable states to get the next DFA $\mathcal{A}_{t+1}$ which is then used to get the next action $a_{t+1}$.

**DFA Encoding:** Given a DFA $\mathcal{A}_t$, we first construct a directed graph $\mathcal{G}_{\mathcal{A}_t} = (\mathcal{V}_{\mathcal{A}_t}, \mathcal{E}_{\mathcal{A}_t})$, where $v_i \in \mathcal{V}_{\mathcal{A}_t}$ represents the states of $\mathcal{A}_t$ and $e_{vu} = (v, u, \mathcal{F}_{vu}) \in \mathcal{E}_{\mathcal{A}_t}$ represents the transitions of $\mathcal{A}_t$. Each edge $e_{vu}$ contains a propositional logic formula $\mathcal{F}_{vu}$ denoting the condition to take the transition. Notice that we use propositional logic formulas for transition conditions rather than individual input symbols. The correspondence between the two DFA definitions is trivial and therefore omitted.

Using the graph representation $\mathcal{G}_{\mathcal{A}_t} = (\mathcal{V}_{\mathcal{A}_t}, \mathcal{E}_{\mathcal{A}_t})$ of $\mathcal{A}_t$, we then construct the DFA encoding $\hat{\mathcal{G}}_{\mathcal{A}_t} = (\hat{\mathcal{V}}_{\mathcal{A}_t}, \hat{\mathcal{E}}_{\mathcal{A}_t}, \hat{\mathcal{R}}_{\mathcal{A}_t})$, where $\hat{\mathcal{V}}_{\mathcal{A}_t}$ is the set of nodes, $\hat{\mathcal{E}}_{\mathcal{A}_t}$ is the set of edges, and $\hat{\mathcal{R}}_{\mathcal{A}_t}$ is the set of edge types. We sketch the iterative the construction of $\hat{\mathcal{G}}_{\mathcal{A}_t}$ below.

Initially, $\hat{\mathcal{G}}_{\mathcal{A}_t}$ has all the nodes of $\mathcal{G}_{\mathcal{A}_t}$. For each $e_{vu} = (v, u, \mathcal{F}_{vu}) \in \mathcal{E}_{\mathcal{A}_t}$, enumerate all *models* (satisfying truth value assignments to atomic propositions) of $\mathcal{F}_{vu}$. For each model, add a new node to $\hat{\mathcal{V}}_{\mathcal{A}_t}$ and two new edges to $\hat{\mathcal{E}}_{\mathcal{A}_t}$ – one from $v$ to the new node (edge type 2) and another one from

the new node to $u$ (edge type 3). Then, add self-loop edges (edge type 1) for each node. Once $\hat{\mathcal{G}}_{\mathcal{A}_t}$ is constructed this way, we reverse its edges to ensure that the message passing in the GNN transmits information to the initial state of the DFA (cf. Figure 1b). Each node $v \in \hat{\mathcal{V}}_{\mathcal{A}_t}$ is associated with an input node feature $h_v^{(0)}$. For $v \in \mathcal{V}_{\mathcal{A}_t}$, $h_v^{(0)}$ indicates whether $v$ is the initial state and whether it is an accepting/rejecting state. For $v \in \hat{\mathcal{V}}_{\mathcal{A}_t} \setminus \mathcal{V}_{\mathcal{A}_t}$, $h_v^{(0)}$ encodes the model associated with $v$.

**DFA Embedding:** Given a DFA encoding $\hat{\mathcal{G}}_{\mathcal{A}_t} = (\hat{\mathcal{V}}_{\mathcal{A}_t}, \hat{\mathcal{E}}_{\mathcal{A}_t}, \hat{\mathcal{R}}_{\mathcal{A}_t})$ and its input node features $\mathcal{H} = \{h_v^{(0)} \mid \forall v \in \hat{\mathcal{V}}_{\mathcal{A}_t}\}$, we construct an embedding of the DFA using a *relational graph convolutional network* (RGCN) [9]. The RGCN performs a sequence of *message passing* steps to map each node to a vector. At a message passing step $t$, the embedding $h_v^{(t)} \in \mathbb{R}^{d^{(t)}}$ of node $v \in \hat{\mathcal{V}}_{\mathcal{A}_t}$ is updated as:

$$h_v^{(t+1)} = \sigma\left(\sum_{r \in \mathcal{R}} \sum_{u \in \mathcal{N}_{\mathcal{G}}^r(v)} \frac{1}{|\mathcal{N}_{\mathcal{G}}^r(v)|} W_r^{(t)} h_u^{(t)}\right),$$

where $\sigma$ is an activation function and $\mathcal{N}_{\mathcal{G}}^r(v)$ denotes the set of nodes adjacent to $v$ via an edge of type $r \in \mathcal{R}$. Notice that weights are shared only for edges of the same type. We perform $8$ message passing steps and 32-dimensional node embeddings, i.e., $h_v^{(t)} \in \mathbb{R}^{32}$, and the input node features are represented by 22-dimensional node embeddings, i.e., $h_v^{(0)} \in \mathbb{R}^{22}$. We use Tanh as the activation function. The feature vector of the DFA's initial state after message passing then represents the latent space encoding $\overrightarrow{\mathcal{A}_t}$ of the DFA task $\mathcal{A}_t$, and it is concatenated with the observation embedding before being passed to the feed-forward policy to compute the likelihood over actions.

## 3 Experience Replay with Automata

Our training was done using an entropy regularized [4] variant of DQN [8] and a replay buffer storing past experiences. Asynchronously to the automata conditioned RL process described in Section 2, we also perform experience replay [2]. That is, we sample episodes from the replay buffer and, using a *relabeling strategy*, relabel these episodes with a new DFA that is satisfied over the sampled episode. These relabeled episodes are then placed back into the replay buffer with their relabeled DFA goal. The relabeling strategy can be a strategy specific to the class of tasks under consideration or a general approach based on *inverse reinforcement learning* [1] as suggested by [3]. In this paper, we focus on two task classes: *reach-avoid tasks* and *reach tasks*, and we implement relabeling strategies for them.

In its simplest form, a reach-avoid task specifies an atomic proposition that has to be satisfied and another one that has to be avoided to accomplish the task, e.g., the DFA in Figure 1b is a reach-avoid task specifying that the agent needs to go to a blue square while avoiding red squares in Figure 1c. Such simple reach-avoid tasks can be combined to express a *sequence of reach-avoid tasks*. Moreover, a path through an *arbitrary* DFA task can be considered as a sequence of reach-avoid tasks since, to realize a given state transition, there is a set of atomic propositions that have to be satisfied and another set of atomic propositions that have to be avoided. The specific reach-avoid task class we study specifies a sequence of $n$ atomic propositions that have to be satisfied in the given order to achieve the goal[1] and $k$ atomic propositions that have to be avoided at each state. For example, a reach-avoid task with $n = 2$ and $k = 1$: go to a blue square while avoiding red squares and then go to a yellow square while avoiding blue squares in Figure 1c. For this class of tasks, we specifically designed a relabeling strategy. Given an episode, we randomly choose $n$ of the atomic propositions that the episode happened to satisfy and construct a DFA specifying the sampled sequence, i.e., starting from the initial state, the only way to get to the accepting state is to satisfy the sampled atomic propositions in the correct order. We then add a rejecting sink state, and for every other state of the DFA, we sample $k$ atomic propositions that are not in the reach set of that state and add a transition to the rejecting sink state conditioned on the sampled $k$ atomic propositions. Finally, we relabel the given episode with this new DFA. Reach tasks are simply reach-avoid tasks with $k = 0$ and their relabeling strategy is identical to reach-avoid, except no rejecting transitions are sampled.

[1] As $n$ grows, the likelihood that a uniformly random policy will satisfy the task shrinks exponentially.

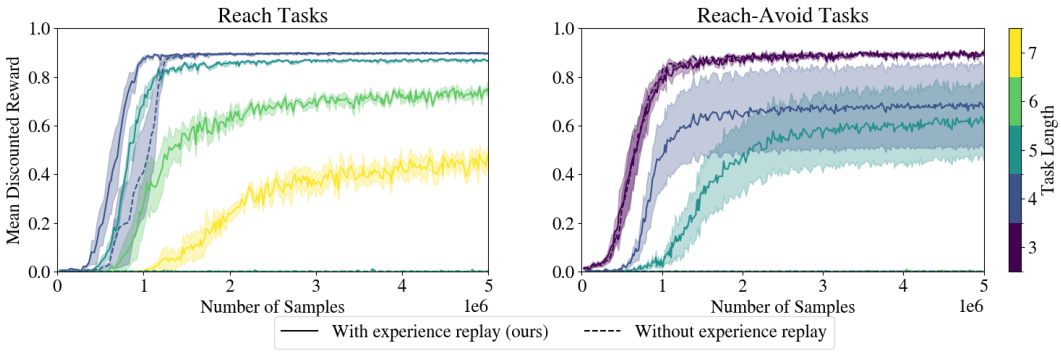



Figure 2: Results show that experience replay helps the RL agent learn more quickly (or, in some cases, learn anything at all), with more gains when the DFA task becomes harder to satisfy. Solid lines use experience replay, and dashed lines do not. Line colors indicate the length of each task, i.e., how many propositions must be seen in sequence to accomplish the task. Reach-avoid tasks need to avoid two propositions (varying at each state in the DFA). The version without experience replay makes no progress for reach tasks past length 4 and the reach-avoid tasks past length 3. Lines are averaged over five samples, and error bars show standard error.

## 4 Experiments[2]

We restrict our experiments to reach and reach-avoid tasks. The underlying environment of our experiments is a 7x7 gridworld depicted in Figure 1c. At each state, the robot occupies a single square of the environment and can move in any of the four cardinal directions. Some squares in the gridworld are randomly colored, and each color corresponds to an atomic proposition. Moving into one of these colored squares satisfies the associated atomic proposition, potentially transitioning the underlying DFA task. At the beginning of each episode, the squares in the environment are randomly colored, and a DFA task from the class is sampled. To satisfy the task and receive a positive reward, the agent must traverse the environment in a way that transitions the DFA into an accepting state within a horizon of 20 steps; otherwise, the agent does not get a reward, i.e., it receives a reward of 0.

For the entropy regularized DQN, we use an exploration parameter that linearly decreases from 1.0 at the first step to 0.05 at step one million. During training, approximately 10% of a batch is relabeled samples. The relabeled and non-relabeled samples are stored in separate buffers of sizes 50,000.

Our experimental results show that experience replay makes a significant difference in how quickly goal-conditioned RL agents learn for DFA tasks. The more difficult the DFA task, the greater the impact of experience replay. Figure 2 (left) shows the learning curves for reach tasks and Figure 2 (right) shows the learning curves for reach-avoid tasks in the gridworld environment. All curves are averaged over 5 random seeds, and the shaded regions represent standard error. Solid lines use experience replay, and dashed lines do not. With experience replay, our RL agents are able to learn to satisfy length 7 reach tasks within a couple of million steps. Without experience replay the RL agents struggle with anything past length 4 tasks. At length 5 tasks, not using experience replay results in high variance due to the sparse reward signal. By contrast, all our RL agents with experience replay successfully learn their tasks and exhibit low variance, i.e., learning consistency.

## 5 Conclusion and Future Work

This work explored the problem of DFA-conditioned RL a class of sparse, binary, history dependent rewards. To condition the policy on the DFA, we leveraged a graph neural network (GNN) to encode the DFA goal before passing it to a forward policy network. Our experimental results demonstrated the efficacy of the model architecture and highlighted the significant role of experience replay in enhancing the learning speed and reducing the variance of RL agents for DFA tasks. Planned future work includes expanding the class of DFAs in the experiments, leveraging IRL-informed relabeling strategies, supporting decompositions of DFAs, and deploying in higher dimensional environments.

[2]The source code is available at https://github.com/beyazit-yalcinkaya/deep-diss.

## References


[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, volume 69 of *ACM International Conference Proceeding Series*. ACM, 2004. 3

[2] Marcin Andrychowicz, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *NIPS*, pages 5048–5058, 2017. 2, 3

[3] Ben Eysenbach, Xinyang Geng, Sergey Levine, and Ruslan Salakhutdinov. Rewriting history with inverse RL: hindsight inference for policy improvement. In *NeurIPS*, 2020. 3

[4] Matthieu Geist, Bruno Scherrer, and Olivier Pietquin. A theory of regularized markov decision processes. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 2160–2169. PMLR, 2019. 3

[5] Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, pages 854–860. IJCAI/AAAI, 2013. 1

[6] Rodrigo Toro Icarte. *Reward Machines*. PhD thesis, University of Toronto, Canada, 2022. 2

[7] Kishor Jothimurugan, Rajeev Alur, and Osbert Bastani. A composable specification language for reinforcement learning tasks. In *NeurIPS*, pages 13021–13030, 2019. 1

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. 3

[9] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15*, pages 593–607. Springer, 2018. 3

[10] Pashootan Vaezipoor, Andrew C. Li, Rodrigo Toro Icarte, and Sheila A. McIlraith. Ltl2action: Generalizing LTL instructions for multi-task RL. In *ICML*, volume 139 of *Proceedings of Machine Learning Research*, pages 10497–10508. PMLR, 2021. 1, 2

[11] Marcell Vazquez-Chanlatte. *Specifications from Demonstrations: Learning, Teaching, and Control*. PhD thesis, University of California, Berkeley, USA, 2022. 1