Automata Guided Reinforcement Learning With Demonstrations

Xiao Li, Yao Ma and Calin Belta

***Abstract*— Tasks with complex structures and long horizons pose a challenge for reinforcement learning agents due to the difficulty in specifying the task in terms of reward functions as well as large variances in the learning signal. We propose to address these problems by combining temporal logic (TL) with reinforcement learning from demonstrations. Our method automatically generates intrinsic rewards that align with the overall task goal given a TL task specification. The policy resulting from our framework has an interpretable and hierarchical structure. We validate the proposed method experimentally on a set of robotic manipulation tasks.**

I. Introduction

Learning robotic skills for tasks with complex structures and long horizons poses a significant challenge for current reinforcement learning methods. Recent endeavors have focused mainly on lower level motor control tasks such as grasping [1] [2], dexterous hand manipulation[3], lego insertion [4]. However, demonstration of robotic systems capable of learning controls for tasks that require logical execution of subtasks has been less successful.

The first challenge in learning of complex tasks is low initial success rates. The agent rarely receives a positive reward signal through exploration. It has been shown that providing demonstration data can significantly facilitate learning. This idea has been adopted to learn tasks such as block stacking [5][6], insertion [7][8] as well as autonomous driving [9][10]. Intrinsic rewards have also been shown to provide extra learning signal [11], however, carelessly designed intrinsic rewards can adversely effect learning performance.

Learning only from demonstrations suffers from covariate shift (accumulative error resulting from deviation of state and action distributions from demonstrations) which is often addressed by combining demonstrations with reinforcement learning [12][13]. However, tasks that require long sequences of actions to complete usually results in high variance learning signals (gradients) which drastically hinders the learning progress. This problem can be alleviated by using temporal abstractions [14]. Hierarchical reinforcement learning has recently been successfully applied in simulated control tasks [15], simple navigation tasks [16] and robotic manipulation tasks [17][18].

The third challenge in learning complex tasks arises with the specification of task rewards. Reward engineering is time-consuming for low-level control tasks where efforts in reward shaping [20] and tuning are often necessary. This process is much more difficult when the structure of the tasks complicates. Authors of [21] have shown that it is already a considerable effort to specify the reward function for simple block-stacking tasks.

X. Li, Y. Ma and C. Belta are with the Dept. of Mechanical Engineering, Boston University, Boston, MA, USA. {xli87, yaoma, cbelta}@bu.edu

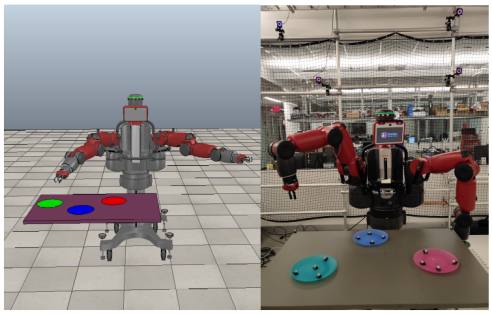

Fig. 1: left: Training environment in the V-REP simulator [19]. **right**: Experimental environment.

We propose to address the above problems by using formal specification languages, particularly temporal logic (TL) as the task specification language. TL has been used in control synthesis [22], path planning [23] and learning [24][25]. It has been shown to provide convenience and performance guarantees in tasks with logical structures and persistence requirements.

Our goal in this work is to provide a framework that integrates temporal logic with reinforcement learning from demonstrations. We show that our framework generates intrinsic rewards that are aligned with the task goals and results in a policy with interpretable hierarchy. We experimentally validate our framework on learning of robotic manipulation tasks with logical structures. All of our training is done in the simulation environment as shown in Figure 1. We show that when configured properly, the policies can transfer directly to the real robot.

II. Related Work

Policy/task sketches have been used to decompose a complex task to a set of sub-tasks [26][27]. However, these methods only support sequential execution of the subtasks whereas our approach is able to compose subtasks in any temporal/logical relationships. Moreover, given the specification of the task in syntactically co-safe truncated linear temporal logic (scTLTL), our method does not require specification of each subtask in terms of reward functions.

The works in [28][29] are the most related to ours. Authors of [28] incorporates maximum-likelihood inverse reinforcement learning with side information (addition constraints of the task) in the form of co-safe linear temporal logic (which is transformed to an equivalent finite state automaton). However, their methods only support discrete state and action

spaces. The authors of [29] propose the reward machine which in effect is an FSA. However, the user is required to manually design the reward machine whereas our method generates the reward machine from TL specifications.

III. Preliminaries

A. Off-Policy Reinforcement Learning

We start with the definition of a Markov Decision Process.

Definition 1: An MDP is defined as a tuple $\mathcal{M} = \langle S, A, p(\cdot|\cdot,\cdot), r(\cdot,\cdot,\cdot)\rangle$, where $S \subseteq \mathbb{R}^n$ is the state space ; $A \subseteq \mathbb{R}^m$ is the action space (S and A can also be discrete sets); $p : S \times A \times S \to [0, 1]$ is the transition function with $p(s'|s, a)$ being the conditional probability density of taking action $a \in A$ at state $s \in S$ and ending up in state $s' \in S$; $r : S \times A \times S \to \mathbb{R}$ is the reward function with $r(s, a, s')$ being the reward obtained by executing action a at state s and transitioning to s'.

We define a task to be the process of finding the optimal policy $\pi^\star : S \to A$ (or $\pi^\star : S \times A \to [0, 1]$ for stochastic policies) that maximizes the expected return, i.e.

$$\pi^\star = \arg\max_{\pi} \mathbb{E}^\pi[\sum_{t=0}^{T-1} r(s_t, a_t, s_{t+1})], \tag{1}$$

The horizon of a task (denoted T) is defined as the maximum allowable time-steps of each execution of π and hence the maximum length of a trajectory. In Equation (1), $\mathbb{E}^\pi[\cdot]$ is the expectation following π. The state-action value function is defined as

$$Q^\pi(s, a) = \mathbb{E}^\pi[\sum_{t=0}^{T-1} r(s_t, a_t, s_{t+1})|s_0 = s, a_0 = a] \tag{2}$$

i.e. it is the expected return of choosing action a at state s and following π onwards. For off-policy actor critic methods such as deep deterministic policy gradient [30], Q^π is used to evaluate the quality of policy π. Parameterized Q^π_w and π_θ (w and θ are learnable parameters) are optimized alternately to obtain $\pi^\star_\theta$.

B. scTLTL and Finite State Automata

We consider tasks specified with *syntactically co-safe Truncated Linear Temporal Logic* (scTLTL) which is derived from truncated linear temporal logic(TLTL) [31]. The $\Box$ (always) operator is omitted in order to establish a connection between TLTL and finite state automaton (Definition 2). The syntax of scTLTL is defined as

$$\begin{aligned}\phi := \top \mid f(s) < c \mid \neg\phi \mid \phi \wedge \psi \mid \\ \Diamond\phi \mid \phi\,\mathcal{U}\,\psi \mid \phi\,\mathcal{T}\,\psi \mid \bigcirc\phi\end{aligned} \tag{3}$$

where $\top$ is the True Boolean constant. $s \in S$ is a MDP state in Definition 1. $f(s) < c$ is a predicate over the MDP states where $c \in \mathbb{R}$. $\neg$ (negation/not), $\wedge$ (conjunction/and) are Boolean connectives. $\Diamond$ (eventually), $\mathcal{U}$ (until), $\mathcal{T}$ (then), $\bigcirc$ (next), are temporal operators.$\Rightarrow$ (implication) and and $\vee$ (disjunction/or) can be derived from the above operators.

We denote $s_t \in S$ to be the state at time t, and $s_{t:t+k}$ to be a sequence of states (state trajectory) from time t to $t+k$, i.e., $s_{t:t+k} = s_t s_{t+1} ... s_{t+k}$. The Boolean semantics of scTLTL is defined as:

$$\begin{aligned}
s_{t:t+k} \models f(s) < c \quad &\Leftrightarrow \quad f(s_t) < c, \\
s_{t:t+k} \models \neg\phi \quad &\Leftrightarrow \quad \neg(s_{t:t+k} \models \phi), \\
s_{t:t+k} \models \phi \Rightarrow \psi \quad &\Leftrightarrow \quad (s_{t:t+k} \models \phi) \Rightarrow (s_{t:t+k} \models \psi), \\
s_{t:t+k} \models \phi \wedge \psi \quad &\Leftrightarrow \quad (s_{t:t+k} \models \phi) \wedge (s_{t:t+k} \models \psi), \\
s_{t:t+k} \models \phi \vee \psi \quad &\Leftrightarrow \quad (s_{t:t+k} \models \phi) \vee (s_{t:t+k} \models \psi), \\
s_{t:t+k} \models \bigcirc\phi \quad &\Leftrightarrow \quad (s_{t+1:t+k} \models \phi) \wedge (k > 0), \\
s_{t:t+k} \models \Diamond\phi \quad &\Leftrightarrow \quad \exists t' \in [t, t+k)\ s_{t':t+k} \models \phi, \\
s_{t:t+k} \models \phi\,\mathcal{U}\,\psi \quad &\Leftrightarrow \quad \exists t' \in [t, t+k)\ s.t.\ s_{t':t+k} \models \psi \\
&\qquad \wedge (\forall t'' \in [t, t')\ s_{t'':t'} \models \phi), \\
s_{t:t+k} \models \phi\,\mathcal{T}\,\psi \quad &\Leftrightarrow \quad \exists t' \in [t, t+k)\ s.t.\ s_{t':t+k} \models \psi \\
&\qquad \wedge (\exists t'' \in [t, t')\ s_{t'':t'} \models \phi).
\end{aligned}$$

A trajectory $s_{0:T}$ is said to satisfy formula ϕ if $s_{0:T} \models \phi$.

There exists a real-valued function $\rho(s_{0:T}, \phi)$ called robustness degree (sometimes referred to as just robustness) that measures the level of satisfaction of trajectory $s_{0:T}$ with respect to a scTLTL formula ϕ. The robustness can be defined recursively as

$$\begin{aligned}
\rho(s_{t:t+k}, \top) &= \rho_{max}, \\
\rho(s_{t:t+k}, f(s_t) < c) &= c - f(s_t), \\
\rho(s_{t:t+k}, \neg\phi) &= -\rho(s_{t:t+k}, \phi), \\
\rho(s_{t:t+k}, \phi \Rightarrow \psi) &= \max(-\rho(s_{t:t+k}, \phi), \rho(s_{t:t+k}, \psi)) \\
\rho(s_{t:t+k}, \phi_1 \wedge \phi_2) &= \min(\rho(s_{t:t+k}, \phi_1), \rho(s_{t:t+k}, \phi_2)), \\
\rho(s_{t:t+k}, \phi_1 \vee \phi_2) &= \max(\rho(s_{t:t+k}, \phi_1), \rho(s_{t:t+k}, \phi_2)), \\
\rho(s_{t:t+k}, \bigcirc\phi) &= \rho(s_{t+1:t+k}, \phi)\ (k > 0), \\
\rho(s_{t:t+k}, \Diamond\phi) &= \max_{t' \in [t, t+k)} (\rho(s_{t':t+k}, \phi)), \\
\rho(s_{t:t+k}, \phi\,\mathcal{U}\,\psi) &= \max_{t' \in [t, t+k)} (\min(\rho(s_{t':t+k}, \psi), \\
&\qquad \min_{t'' \in [t, t')} \rho(s_{t'':t'}, \phi))), \\
\rho(s_{t:t+k}, \phi\,\mathcal{T}\,\psi) &= \max_{t' \in [t, t+k)} (\min(\rho(s_{t':t+k}, \psi), \\
&\qquad \max_{t'' \in [t, t')} \rho(s_{t'':t'}, \phi))),
\end{aligned}$$

where ρ_{max} represents the maximum robustness value. A robustness of greater than zero implies that $s_{t:t+k}$ satisfies ϕ and vice versa ($\rho(s_{t:t+k}, \phi) > 0 \Rightarrow s_{t:t+k} \models \phi$ and $\rho(s_{t:t+k}, \phi) < 0 \Rightarrow s_{t:t+k} \not\models \phi$). The robustness can substitute Boolean semantics to enforce the specification ϕ.

Definition 2: An FSA corresponding to a scTLTL formula ϕ [1] is defined as a tuple $\mathcal{A}_\phi = \langle \mathbb{Q}_\phi, \Psi_\phi, q_0, p_\phi(\cdot|\cdot), \mathcal{F}_\phi\rangle$, where $\mathbb{Q}_\phi$ is a set of automaton states; Ψ_ϕ is the input alphabet (a set of first order logic formula); $q_0 \in \mathbb{Q}_\phi$ is

[1] Here we slightly modify the conventional definition of FSA and incorporate the probabilities in Equations (4). For simplicity, we continue to adopt the term FSA.

the initial state; $p_\phi : \mathbb{Q}_\phi \times \mathbb{Q}_\phi \to [0,1]$ is a conditional probability defined as

$$p_\phi(q_j|q_i) = \begin{cases} 1 & \psi_{q_i,q_j} \text{ is true} \\ 0 & \text{otherwise.} \end{cases}$$
$$or \tag{4}$$
$$p_\phi(q_j|q_i, s) = \begin{cases} 1 & \rho(s, \psi_{q_i,q_j}) > 0 \\ 0 & \text{otherwise.} \end{cases}$$

$\mathcal{F}_\phi$ is a set of final automaton states. The transitions in the FSA are deterministic. For reasons that will become clear later, we adopt the probability notation in Equation (4) so that we can combine it with an MDP transition.

We denote $\psi_{q_i,q_j} \in \Psi_\phi$ the predicate guarding the transition from q_i to q_j. Because ψ_{q_i,q_j} is a predicate without temporal operators, the robustness $\rho(s_{t:t+k}, \psi_{q_i,q_j})$ is only evaluated at s_t. Therefore, we use the shorthand $\rho(s_t, \psi_{q_i,q_j}) = \rho(s_{t:t+k}, \psi_{q_i,q_j})$. The translation from a TLTL formula to a FSA can be done automatically with available packages like Lomap [32]. An example of scTLTL is provided in the next section.

IV. Problem Formulation and Approach

Problem 1: Given an MDP $\mathcal{M} = \langle S, A, p(\cdot|\cdot,\cdot), r(\cdot,\cdot,\cdot)\rangle$ with unknown transition dynamics $p(\cdot|\cdot,\cdot)$ and a scTLTL formula ϕ as in Definition 2, find a policy $\pi^\star_\phi$ such that

$$\pi^\star_\phi = \arg\max_{\pi_\phi} \mathbb{E}^{\pi_\phi}[\mathbb{1}(\rho(s_{0:T}, \phi) > 0)]. \tag{5}$$

where $\mathbb{1}(\rho(s_{0:T}, \phi) > 0)$ is an indicator function with value 1 if $\rho(s_{0:T}, \phi) > 0$ and 0 otherwise.

$\pi^\star_\phi$ in Equation (5) is said to satisfy ϕ. Problem 1 defines a policy search problem where the trajectories resulting from following the optimal policy should satisfy the given scTLTL formula in expectation. On a high level, our approach is to construct a product MDP between $\mathcal{M}$ and $\mathcal{A}_\phi$ and learn policy π_ϕ using the product. To accelerate learning, we provide human demonstrations of the task specified by ϕ and provide a simple technique to transform the demonstrations compatible with the product MDP.

V. FSA Augmented MDP

We introduce the FSA augmented MDP:

Definition 3: An FSA augmented MDP corresponding to scTLTL formula ϕ (constructed from FSA $\langle \mathbb{Q}_\phi, \Psi_\phi, q_0, p_\phi(\cdot|\cdot), \mathcal{F}_\phi\rangle$ and MDP $\langle S, A, p(\cdot|\cdot,\cdot), r(\cdot,\cdot,\cdot)\rangle$) is defined as $\mathcal{M}_\phi = \langle \tilde{S}, A, \tilde{p}(\cdot|\cdot,\cdot), \tilde{r}(\cdot,\cdot), \mathcal{F}_\phi\rangle$ where $\tilde{S} \subseteq S \times \mathbb{Q}_\phi$, $\tilde{p}(\tilde{s}'|\tilde{s}, a)$ is the probability of transitioning to $\tilde{s}'$ given $\tilde{s}$ and a,

$$\tilde{p}(\tilde{s}'|\tilde{s}, a) = p\big((s', q')|(s, q), a\big) = \begin{cases} p(s'|s, a) & p_\phi(q'|q, s) = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

p_ϕ is defined in Equation (4). $\tilde{r} : \tilde{S} \times \tilde{S} \to \mathbb{R}$ is the FSA augmented reward function, defined by

$$\tilde{r}(\tilde{s}, \tilde{s}') = \rho(s', D^q_\phi), \tag{7}$$

where $D^q_\phi = \bigvee_{q' \in \Omega_q} \psi_{q,q'}$ represents the disjunction of all predicates guarding the transitions that originate from q (Ω_q is the set of automata states that are connected with q through outgoing edges). Equation (7) effectively acts as an intrinsic reward that aligns with the overall goal of Equation (5).

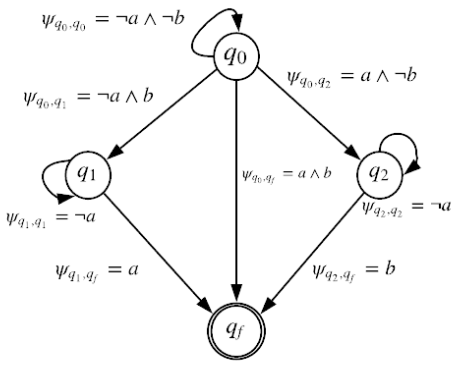

Fig. 2: Finite state automaton generated from formula $\Diamond a \wedge \Diamond b$

Example 1: Figure 2 illustrates the FSA resulting from formula $\phi = \Diamond a \wedge \Diamond b$ (where $a : s > 3 \wedge s < 5, b : s > 8 \wedge s < 10$ are predicates over states). In English, ϕ entails that during a run, regions specified by a and b need to be visited at least once. The FSA has four automaton states $Q_\phi = \{q0, q_1, q_2, q_f\}$ with q_0 being the input(initial) state (here q_i serves to track the progress in satisfying ϕ). The input alphabet is defined as $\Psi_\phi = \{\neg a \wedge \neg b, \neg a \wedge b, a \wedge \neg b, a \wedge b\}$. Shorthands are used in the figure, for example $a = (a \wedge b) \vee (a \wedge \neg b)$. Ψ_ϕ represents the power set of $\{a, b\}$, i.e. $\Psi_\phi = 2^{\{a,b\}}$. During execution, the FSA always starts from state q_0, s_0 and transitions according to Equation (6). The specification is satisfied when q_f is reached.

The goal is to find the optimal policy that maximizes the expected sum of discounted return, i.e.

$$\pi^\star_\phi = \arg\max_{\pi_\phi} \mathbb{E}^{\pi_\phi}\left[\sum_{t=0}^{T-1} \gamma^{t+1} \tilde{r}(\tilde{s}_t, \tilde{s}_{t+1})\right], \tag{8}$$

where $\gamma < 1$ is the discount factor, T is the time horizon.

The reward function in Equation (7) encourages the system to exit the current automaton state and move on to the next, and by doing so eventually reach the final state q_f (property of FSA) which satisfies the TL specification and hence Equation (5). The discount factor in Equation (8) reduces the number of satisfying policies to one.

The FSA augmented MDP can be constructed with any standard MDP and a scTLTL formula, and Equation (8) can be solved with any off-the-shelf RL algorithm. After obtaining the optimal policy $\pi^\star_\phi$, executing $\pi^\star_\phi(s_t, q_i)$ without transitioning the automaton state (i.e. keeping q_i fixed) results in a set of meaningful policies that can be used as is or composed with other such policies.

VI. FSA Guided Reinforcement Learning From Demonstrations

In this section, we introduce our main algorithm - FSA guided reinforcement learning from demonstrations. The algorithm takes as input a scTLTL formula ϕ, a randomly initialized policy $\pi_\theta(s,q)$ and a set of demonstration trajectories $D = \{\tau_i\}, i \in 1,...,n$ that satisfy ϕ, where $\tau = (s_0, a_0, ..., s_T)$ is the state-action trajectory. The algorithm consists of the following steps:

1) Construct the FSA augmented MDP $\mathcal{M}_\phi$.
2) For each demonstration trajectory τ_i, construct the Q-appended demonstration trajectory $\tau_i^Q = (s_0, a_0, q_0, ..., s_T)$ by finding the corresponding q_t for each (s_t, a_t) using Equation (4). Denote $\mathcal{D}^Q = \{\tau_i^Q\}, i = 1, .., n$.
3) Perform behavior cloning (supervise learning on the demonstration trajectories) to initialize policy (details provided in Section VII-B).
4) Train the agent using any reinforcement learning from demonstration algorithm (such as [13], [5], [12]).

Algorithm 1 shows each step with its inputs and output. We will discuss our choices of behavior cloning and RL algorithms in the next section.

Algorithm 1 FSA Guided Reinforcement Learning From Demonstrations

1: **Inputs**: scTLTL task specification ϕ, randomly initialized policy π_θ, a set of n demonstration trajectories $\mathcal{D} = \{\tau_i\}, i \in 1, ..., n$.
2: Construct the FSA augmented MDP $\mathcal{M}_\phi$
3: $\mathcal{D}^Q \leftarrow$ *ConstructQAppendedDemoBatch*$(\mathcal{M}_\phi, \mathcal{D})$
4: $\theta \leftarrow$ *BehaviorCloning*$(\theta, \mathcal{D}^Q)$
5: $\theta^\star \leftarrow$ *RLfD*$(\mathcal{M}_\phi, \mathcal{D}^Q)$ ▷ $RLfD$ stands for any learning from demonstration algorithm

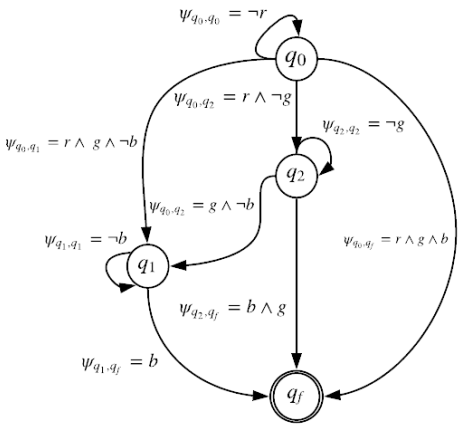

Fig. 3: Finite state automaton generated from formula $\Diamond(r \wedge \Diamond(g \wedge \Diamond b))$.

In this section we present some preliminary experimental results using the FSA augmented MDP to learn temporal logic specified tasks.

VII. Experiments

A. Experiment Setup

As shown in Figure 4, we control one arm of a Baxter robot (7 degrees of freedom) to traverse among three regions defined by the red, green and blue disks. The positions of the disks are tracked by our motion capture system and thus fully observable. Our state space is 16 dimensional that includes 7 joint angles and the three disk positions relative to the gripper (9 dimensional) denoted by $\boldsymbol{p}^{red}, \boldsymbol{p}^{green}, \boldsymbol{p}^{blue}$. Our action space is the 7-dimensional joint velocities. We define three predicates $\psi_i = |\boldsymbol{p}^i| < \epsilon, i \in \{red, green, blue\}$, ϵ is a threshold which we set to be 5 centimeters.

We test our algorithm on two tasks

- Task 1: $\phi_1 = \Diamond(\psi^{red} \wedge \Diamond(\psi^{green} \wedge \Diamond\psi^{blue}))$
  Description: visit regions red, green, and blue in this order.
- Task 2: $\phi_2 = \Diamond\psi^{red} \wedge \Diamond\psi^{green} \wedge \Diamond\psi^{blue}$
  Description: Eventually visit regions red, green and blue. Order does not matter.

Figure 3 shows the FSA resulting from ϕ_1. The FSA for ϕ_2 is similar in nature to that presented in Figure 2 and therefore not included due to space constraints.

B. Algorithm Details

For each task, we collect 50 human demonstration state-action trajectories (each demonstration about 12 seconds long) with randomized initial conditions (arm configuration and position of the regions). Demonstrations are collected by holding Baxter's gripper in gravity compensation mode while performing the task. Behavior cloning is used to initialize the policy with the following loss function

$$L_{BC} = \sum_{i=0}^{N_D} ||\pi_\theta(s_i) - a_i||^2, \tag{9}$$

where $\pi_\theta(s_i) : S \rightarrow A$ is a deterministic policy represented by a feedforward neural network with 3 layers, each lay consisting of 100 relu units. N_D is the number of samples. Other behavior cloning losses can also be used [33].

We use deep deterministic policy gradient (DDPG) [30] as our reinforcement learning algorithm. During training, we maintain two replay buffers, one for interaction data and one for demonstration data. At each update step, we sample a batch of experience from the interaction data buffer using prioritized experience replay [34] and another batch from the demonstration data buffer and combine the two batches for one update. In addition, we modify the policy loss to be

$$L_{total} = L_{DDPG} + \lambda L_{BC}, \tag{10}$$

where L_{DDPG} is the usual DDPG actor loss (similar technique is used in [5]). During training, we linearly decay λ from 0.8 to 0.1 over 30000 update steps to favor demonstration in the beginning and unbiased DDPG loss towards the end (similar technique is used in [12]). We set the horizon T to be 100 steps (5 seconds). 5 episodes of exploration data are collected to perform 10 updates. We use a learning rate

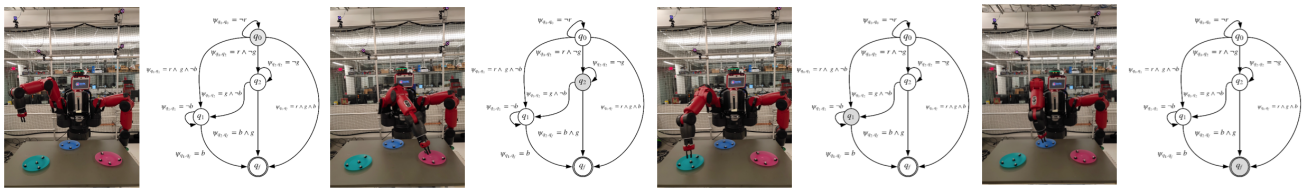

Fig. 4: Sample execution of task 1: $\Diamond(\psi^{red} \wedge \Diamond(\psi^{green} \wedge \Diamond\psi^{blue}))$ with FSA (same as Figure 3) transitions shown. The shaded q state represents the current automaton state.

of 0.0003, a discount factor of 0.99, batch size of 32 (from both buffers).

We randomly initialize the joint angles, the automaton state as well as the positions of the regions at reset of each episode in order to achieve generalization over different configurations of the workspace. An episode resets if the gripper comes too close to the table. All of our training is performed in simulation using the V-REP platform [19]. The simulation environment is calibrated to the real world workspace. We set the control frequencies in both the real and simulated robot to be 20 Hz and show that the learned policies transfer directly to the real robot without fine-tuning.

C. Comparison Cases

As comparison, we introduce a binary vector b with three digits. A digit in b is 1 if the corresponding region has been reached at least once and 0 otherwise (i.e. $b = 100$ if $\epsilon - |\boldsymbol{p}^{red}| > 0$ occurs at least once in an episode. Likewise for $b = 010$ for blue and $b = 001$ for green). b is used to track progress towards accomplishing the task. We train each task with the following shaped reward

$$r_{\phi_1} = \begin{cases} \epsilon - |\boldsymbol{p}^{red}| & b = 000 \\ \epsilon - |\boldsymbol{p}^{green}| & b = 100 \\ \epsilon - |\boldsymbol{p}^{blue}| & b = 110 \\ -2 & \text{otherwise.} \end{cases} \tag{11}$$

$$r_{\phi_2} = \begin{cases} \max(\epsilon - |\boldsymbol{p}^{red}|, \epsilon - |\boldsymbol{p}^{green}|, \epsilon - |\boldsymbol{p}^{blue}|) & b = 000 \\ \max(\epsilon - |\boldsymbol{p}^{green}|, \epsilon - |\boldsymbol{p}^{blue}|) & b = 100 \\ \max(\epsilon - |\boldsymbol{p}^{red}|, \epsilon - |\boldsymbol{p}^{blue}|) & b = 010 \\ \max(\epsilon - |\boldsymbol{p}^{red}|, \epsilon - |\boldsymbol{p}^{green}|) & b = 001 \\ \epsilon - |\boldsymbol{p}^{blue}| & b = 110 \\ \epsilon - |\boldsymbol{p}^{green}| & b = 101 \\ \epsilon - |\boldsymbol{p}^{red}| & b = 011 \\ -2 & \text{otherwise.} \end{cases} \tag{12}$$

on the original MDP. We also compare cases with and without demonstration.

Due to the scale difference between rewards provided by the FSA augmented MDP and the shaped reward, we present all learning curves in terms of robustness for a clear comparison. This is because the semantics of the robustness entails that a trajectory evaluating to a higher robustness value achieves better satisfaction of the TL specification (a value greater than zero guarantees satisfaction).

We acknowledge that for any given task, a well-shaped reward that accelerates learning can be provided if enough effort goes into the design and tuning process. However, this effort grows quickly with the complexity of the task. Our goal is to use formal languages to free users of this burden while achieving similar sample efficiency as a shaped reward.

VIII. Results and Discussion

In this section, we present our experimental results along with discussions of their implications. Figure 4 shows an example execution of task 1 on Baxter. The automaton serves as a progress tracking mechanism that hierarchically abstracts a temporal dependent task to a set of independent ones.

As stated in Section VII-C, since we are training with different reward functions, in order for a fair comparison, we sample a batch of 10 trajectories every 25,000 environmental steps (robot interaction step, as opposed to policy update step) and calculate the robustness for each trajectory. Their means and standard deviations are presented in Figure 5. In the figure, we refer to Algorithm 1 as 'Ours', and learning from only FSA augmented MDP as 'Ours without demonstration'. The shaped rewards are used to train with the same learning procedure as stated in Section VII-B.

The results in Figure 5 show that our method is able to solve both tasks with and without demonstrations (task is considered solved if the average robustness stabilizes above zero). However, demonstrations and behavior cloning significantly decreased the time to convergence as well as the variance during training. We can see that the agent is also able to learn very slowly using the shaped rewards but is unable to solve either task in the allocated time. The speedup of our method is mainly due to the temporal hierarchy the FSA provides. By adding one discrete dimension (the q state) to the state space and randomizing on that dimension during learning, a curriculum is created to help the agent learn a set of simpler sub-tasks building up to the final task. This way the agent is able to visit various states along the task without having to first learn the correct actions leading up to those states.

In all comparison cases, learning task 2 if faster than task 1. This is because task 1 imposes more constraints on the desired behavior (ordering). It is expected that even with the shaped reward, demonstration and behavior cloning is able

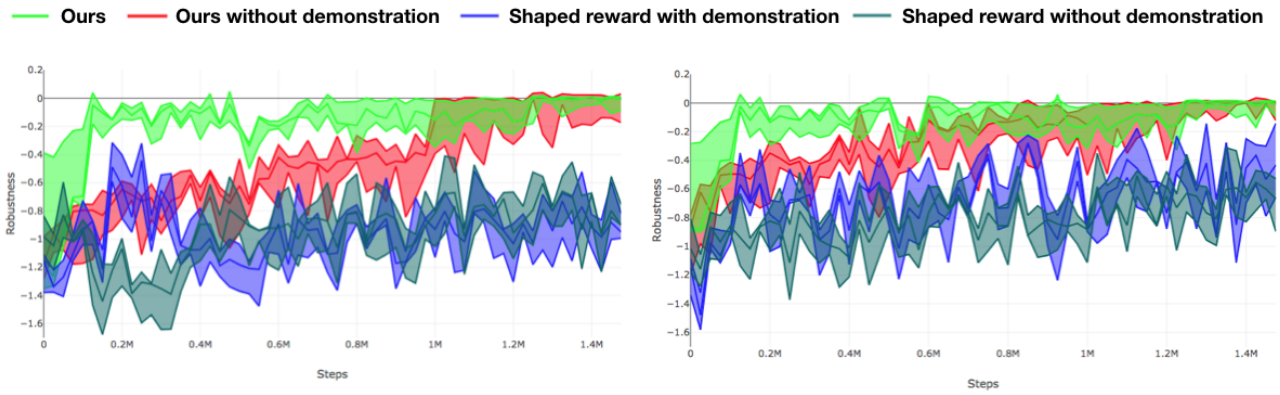

Fig. 5: Learning curve for **left**: Task 1 and **right**: Task 2. Steps here are referred to as environmental step

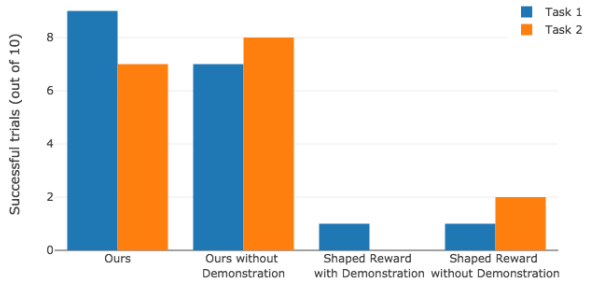

Fig. 6: Task success rate of the trained policies.

TABLE I: Average number of steps to finish the task

	Task 1	Task 2
Ours	36.5	34.3
Ours without demostration	35.7	34.1
Shaped reward with demonstration	100	93.2
Shaped reward without demonstration	99.6	95.5

help bootstrap learning at the initial stages. However, such initialization can be damaged as shown in Figure 5 left. After training, we evaluate the policies by running 10 trials with randomly initialized robot and workspace configurations. Results in Figure 6 show that the resulting policies from our method (with and without demonstrations) is able to accomplish the tasks relatively reliably whereas the policies from the shaped rewards struggled.

It should be noted that there typically will be more than one policy that satisfies Equation (5). However, the discount factor in Equation (8) reduces the number of optimal policies to one (the one that yields a satisfying trajectory in the least number of steps). Table I shows the average number of steps each policy takes to accomplish the corresponding task.

As with any formal method based technique, there is a learning curve to understanding formal languages and using them well in writing specifications. We find that the FSA has significantly helped us in understanding what we are specifying to the agent which served as an effective means to alleviate reward hacking [35]. At its current state, our framework does not support specification of persistent tasks [36]. We have also yet to demonstrate tasks specified over MDP states and actions (e.g. if some state occurs then do something). These are possible extensions of future work.

IX. Conclusions

Learning to follow logical instruction can be useful in real life (e.g. following a recipe or the traffic rules). In this work, we proposed a method to combine temporal logic with reinforcement learning from demonstrations which provides the agent with temporal hierarchy and task aligned intrinsic rewards. We showed that comparing to heuristically designed reward functions, our method provides a formalism for task specification and is able to learn with less experience. By toggling the automaton state q, our learned policy is able to exhibit different behaviors specified by the intrinsic reward in Equation (7) even though no hierarchy is imposed on the policy architecture (simple feedforward neural network). For future work, we will take advantage of this characteristic and develop a set of techniques for skill composition and task-space transfer. We will also demonstrate our methods on more complex tasks.

References

[1] J. Mahler, M. Matl, X. Liu, A. Li, D. V. Gealy, and K. Y. Goldberg, "Dex-net 3.0: Computing robust robot suction grasp targets in point clouds using a new analytic model and deep learning," CoRR, vol. abs/1709.06670, 2017.

[2] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," CoRR, vol. abs/1806.10293, 2018.

[3] A. Marcin, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," arXiv preprint arXiv:1808.00177, 2018.

[4] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, and S. Levine, "Composable deep reinforcement learning for robotic manipulation," CoRR, vol. abs/1803.06773, 2018.

[5] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," CoRR, vol. abs/1709.10089, 2017.

[6] Y. Zhu, Z. Wang, J. Merel, A. A. Rusu, T. Erez, S. Cabi, S. Tunyasuvunakool, J. Kramár, R. Hadsell, N. de Freitas, and N. Heess, "Reinforcement and imitation learning for diverse visuomotor skills," CoRR, vol. abs/1802.09564, 2018.

[7] H.-C. Lin, T. Tang, Y. Fan, Y. Zhao, M. Tomizuka, and W. Chen, "Robot learning from human demonstration with remote lead hrough teaching," 2016 European Control Conference (ECC), pp. 388–394, 2016.

[8] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. A. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," CoRR, vol. abs/1707.08817, 2017.

[9] D. Silver, J. A. Bagnell, and A. Stentz, "Learning autonomous driving styles and maneuvers from expert demonstration," in ISER, 2012.

[10] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. A. Theodorou, and B. Boots, "Agile autonomous driving using end-to-end deep imitation learning," 2010.

[11] S. P. Singh, A. G. Barto, and N. Chentanez, "Intrinsically motivated reinforcement learning," in NIPS, 2004.

[12] A. Rajeswaran, V. Kumar, A. Gupta, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations."

[13] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," CoRR, vol. abs/1804.02717, 2018.

[14] R. S. Suttona, D. Precupb, and S. Singha, "Between mdps and semi-mdps : A framework for temporal abstraction in reinforcement learning," 1998.

[15] O. Nachum, S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," CoRR, vol. abs/1805.08296, 2018.

[16] B. Bischoff, D. Nguyen-Tuong, I.-H. Lee, F. Streichert, and A. Knoll, "Hierarchical reinforcement learning for robot navigation," in ESANN, 2013.

[17] A. Gudimella, R. Story, M. Shaker, R. Kong, M. Brown, V. Shnayder, and M. Campos, "Deep reinforcement learning for dexterous manipulation with concept networks," CoRR, vol. abs/1709.06977, 2017.

[18] D. Xu, S. Nair, Y. Zhu, J. Gao, A. Garg, L. Fei-Fei, and S. Savarese, "Neural task programming: Learning to generalize across hierarchical tasks," CoRR, vol. abs/1710.01813, 2017.

[19] E. Rohmer, S. P. N. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1321–1326, 2013.

[20] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in ICML, 1999.

[21] I. Popov, N. Heess, T. P. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. A. Riedmiller, "Data-efficient deep reinforcement learning for dexterous manipulation," CoRR, vol. abs/1704.03073, 2017.

[22] S. Andersson, A. Nikou, and D. V. Dimarogonas, "Control synthesis for multi-agent systems under metric interval temporal logic specifications," IFAC-PapersOnLine, vol. 50, no. 1, pp. 2397–2402, 2017.

[23] K. J. Leahy, D. Aksaray, and C. Belta, "Informative path planning under temporal logic constraints with performance guarantees," in American Control Conference (ACC), 2017. IEEE, 2017, pp. 1859–1865.

[24] Y. Chen, J. Tumova, and C. Belta, "Ltl robot motion control based on automata learning of environmental dynamics," 2012 IEEE International Conference on Robotics and Automation, pp. 5177–5182, 2012.

[25] D. Kasenberg and M. Scheutz, "Interpretable apprenticeship learning with temporal logic specifications," in Decision and Control (CDC), 2017 IEEE 56th Annual Conference on. IEEE, 2017, pp. 4914–4921.

[26] J. Andreas, D. Klein, and S. Levine, "Modular multitask reinforcement learning with policy sketches," in ICML, 2017.

[27] K. Shiarlis, M. Wulfmeier, S. Salter, S. Whiteson, and I. Posner, "Taco: Learning task decomposition via temporal alignment for control," CoRR, vol. abs/1803.01840, 2018.

[28] M. Wen, I. Papusha, and U. Topcu, "Learning from demonstrations with high-level side information," in IJCAI, 2017.

[29] R. T. Icarte, T. Klassen, R. Valenzano, and S. McIlraith, "Using reward machines for high-level task specification and decomposition in reinforcement learning," in International Conference on Machine Learning, 2018, pp. 2112–2121.

[30] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," CoRR, vol. abs/1509.02971, 2015.

[31] X. Li, C.-I. Vasile, and C. Belta, "Reinforcement learning with temporal logic rewards," in 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2017, pp. 3834–3839.

[32] C. Vasile, Github repository, 2017.

[33] A. A. Rusu, S. G. Colmenarejo, aglar Gülehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell, "Policy distillation," CoRR, vol. abs/1511.06295, 2015.

[34] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," CoRR, vol. abs/1511.05952, 2015.

[35] D. Amodei, C. Olah, J. Steinhardt, P. F. Christiano, J. Schulman, and D. Mané, "Concrete problems in ai safety," CoRR, vol. abs/1606.06565, 2016.

[36] S. L. Smith, J. Tumova, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," I. J. Robotics Res., vol. 30, pp. 1695–1708, 2011.