Automata Guided Reinforcement Learning And Skill Composition

Author Names Omitted for Anonymous Review. Paper-ID [add your ID here]

***Abstract*—Tasks with complex structures and long horizons pose a challenge for reinforcement learning (RL) agents. Skills learned through RL often generalize poorly across tasks and re-training is usually necessary. We propose a framework that combines temporal logic (TL) with reinforcement learning to address these problems. We also present a technique that constructs new skills from existing ones with no additional exploration. Our method allows for convenient specification of complex temporal dependent tasks using logical expressions and automatically generates rewards that align with the overall task goal. We provide theoretical results for our method and evaluate it in both simulation and experiments.**

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 on lower level motor control tasks such as grasping [17] [13], dexterous hand manipulation[18], lego insertion [9]. However, demonstration of robotic systems capable of learning controls for tasks that require logical execution of subtasks has been less explored.

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. The authors of [22] have shown that it is already a considerable effort to specify the reward function for block-stacking tasks.

Policies learned using reinforcement learning aim to maximize the given reward function and are often difficult to transfer to other problem domains. Skill composition is the process of constructing new skills out of existing ones (policies) with little to no additional learning. In stochastic optimal control, this idea has been adopted by [27] and [5] to construct provably optimal control laws based on linearly solvable Markov decision processes.

We propose to address the above problems by using formal languages for task specification, particularly syntactically co-safe truncated linear temporal logic (scTLTL). We take advantage of the equivalence between an scTLTL formula and a finite state automaton (FSA) to construct a reward that aligns with the specification. We also use the FSA to achieve skill composition. Compared to most heuristic reward structures used in the RL literature, our formal specification language has the advantage of semantic rigor and interpretability.

Our contributions are as follows:

- we provide a framework that integrates temporal logic with reinforcement learning. We show that our framework generates rewards that are aligned with the task goals and results in a policy with interpretable hierarchy.
- we introduce a method to compose policies learned using the above framework. We build on the results of [29] and prove that the composed policy is optimal in both $-AND-$ and $-OR-$ task compositions. We show that incorporating temporal logic allows us to compose tasks of greater logical complexity.
- we evaluate our method in simulation (discrete state and action spaces) and experimentally on a Baxter robot (continuous state and action spaces).

II. Related Work

Using temporal logic in a RL setting has been explored in the past. The authors of [7] and [4] use temporal logic and automata to solve the non-Markovian reward decision process (NMRD). In [1], the authors take advantage of the robustness degree of signal temporal logic to guide the learning process. The authors of [32] incorporate maximum-likelihood inverse reinforcement learning with addition constraints of the task in the form of co-safe linear temporal logic. However, their method only supports discrete state and action spaces. In [12], the authors propose a reward machine, which in effect is an FSA. However, the user is required to manually design the reward machine. Our method generates the reward machine directly from TL specifications. In [2], the authors use an FSA as a safety measure that supervises the actions sent to the system by the agent.

Hierarchical RL has been used to increase sample efficiency as well as task generalization [26]. Even though our method does not explicitly assume a hierarchical setup or requires a hierarchical RL algorithm to train, combining scTLTL with MDP naturally results in a hierarchical policy that can be trained end-to-end with any RL algorithm.

Recent efforts in skill composition have mainly adopted the approach of combining value functions learned using different rewards. [21] constructs a composite policy by combining the value functions of individual policies using the Boltzmann distribution. With a similar goal, [33] achieves task space transfer using deep successor representations [14]. However, it is required that the reward function be represented as a linear combination of state-action features. The authors of [3] use olicy sketches for composition. However, only sequential subtask execution is supported.

The authors of [10] have showed that when using energy-based models [8], an approximately optimal composite policy can result from taking the average of the Q-functions of

existing policies. The resulting composite policy achieves the $-AND-$ task composition i.e. the composite policy maximizes the average reward of individual tasks. In [29], the authors took this idea a step further and showed that by combining individual Q-functions using the log-sum-exponential function, the $-OR-$ task composition (the composite policy maximizes the (soft) maximum of the reward of constituent tasks) can be achieved optimally.

Multi-task learning [3] and meta-learning [6] are often used to achieve few-shot/zero-shot task generalization. Here we make the distinction between skill composition (our focus) and multi-task learning/meta-learning where the former constructs new policies from a library of learned policies and the latter often learns and generalizes from a predefined set of tasks/task distributions. Contrasting with multi-task/meta-learning is not within the scope of this paper.

In our framework, skill composition is accomplished by taking the product of the finite state automata (FSA). Instead of interpolating/extrapolating among learned skills/latent features [21][33] , our method is based on graph manipulation of the FSAs. Therefore, the outcome is more transparent. Compared with previous work on skill composition, we impose no constraints on the policy representation or the problem class. To the best of our knowledge, this paper presents the first effort in using temporal logic for skill composition and demonstrates its applicability on real robotic systems.

III. Preliminaries

In this section, we start by introducing the entropy-regularized reinforcement learning and its relevant results on policy composition. Then we introduce scTLTL using simple examples along with the definition of the FSA.

A. Entropy-Regularized Reinforcement Learning and Q-Composition

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'.*

In entropy-regularized reinforcement learning [25], the goal is to maximize the following objective

$$J(\pi) = \sum_{t=0}^{T-1} \mathbb{E}^{\pi}[r_t + \alpha\mathcal{H}(\pi(\cdot|s_t))], \tag{1}$$

where $\pi : S \times A \to [0, 1]$ is a stochastic policy. $\mathbb{E}^{\pi}$ is the expectation following π. $\mathcal{H}(\pi(\cdot|s_t))$ is the entropy of π. α is the temperature parameter. In the limit $\alpha \to 0$, Equation (1) becomes the standard RL objective. The soft Q-learning algorithm introduced by [8] optimizes the above objective and finds a policy represented by an energy-based model

$$\pi^{\star}(a_t|s_t) \propto \exp(-\mathcal{E}(s_t, a_t)) \tag{2}$$

where $\mathcal{E}(s_t, a_t)$ is an energy function that can be represented by a function approximator.

Let $r_t^{ent} = r_t + \alpha\mathcal{H}(\pi(\cdot|s_t))$, the state-action value function (Q-function) following π is defined as $Q^{\pi}(s, a) = \mathbb{E}^{\pi}[\sum_{t=0}^{T-1} r_t^{ent}|s_0 = s, a_0 = a]$. Suppose we have a set of n tasks indexed by $i, i \in \{0, ..., n\}$, each task is defined by an MDP $\mathcal{M}_i$ that differs only in their reward function r_i. Let $Q_{\alpha}^{\pi_i^{\star}}$ be the optimal entropy-regularized Q-function.

Theorem 1. *[29] $\boldsymbol{r} = \{r_1, ..., r_n\}$ as a set of rewards, $\boldsymbol{Q}_{\alpha}^{\pi^{\star}} = \{Q_{\alpha}^{\pi_1^{\star}}, ..., Q_{\alpha}^{\pi_n^{\star}}\}$ is set of Q-functions. Given a set of non-negative weights $\boldsymbol{w}$ with $||\boldsymbol{w}|| = 1$, the optimal Q-function for a new task defined by $r = \alpha \log(|| \exp(\boldsymbol{r}/\alpha)||_{\boldsymbol{w}})$ is given by*

$$Q_{\alpha}^{\pi^{\star}} = \alpha \log(|| \exp(\boldsymbol{Q}_{\alpha}^{\pi^{\star}}/\alpha)||_{\boldsymbol{w}}), \tag{3}$$

where $|| \cdot ||_{\boldsymbol{w}}$ is the weighted 1-norm.

Corollary 1. *[29] $\max \boldsymbol{Q}_{\alpha}^{\pi^{\star}} \uparrow Q_0^{\pi^{\star}}$ as $\alpha \to 0$, where $Q_0^{\pi^{\star}}$ is the optimal Q-function for the objective $J(\pi) = \sum_{t=0}^{T-1} \mathbb{E}^{\pi}[r_t]$.*

Corollary 1 states that in the low temperature limit, the maximum of the optimal entropy-regularized Q-functions approaches the standard optimal Q-function. This means as $\alpha \to 0$, $Q_{\alpha}^{\pi^{\star}} = \max(\boldsymbol{Q}_{\alpha}^{\phi^{\star}})$ (The log-sum-exponential expression in Equation (3) is an approximation of the maximum function when $\alpha > 0$). In addition, Equation (1) simply reduces to the expected sum of rewards in this case.

B. scTLTL and Finite State Automata

We consider tasks specified with *syntactically co-safe Truncated Linear Temporal Logic* (scTLTL) which is a restricted version of truncated linear temporal logic(TLTL) [15]. In particular, we restrict from using the $\square$ (always) operator. By doing so, we can establish a correspondence between an scTLTL formula with a FSA.

Due to space constraints, we do not provide the complete set of definitions for the Boolean and quantitative semantics of scTLTL (refer to [15]). Examples of scTLTL include $\Diamond(\phi_a \wedge \Diamond\phi_b)$ which entails that *eventually* ϕ_a and then *eventually* ϕ_b become true (sequencing). Another example $(\phi_a \Rightarrow \Diamond\phi_b)\ \mathcal{U}\ \phi_c$ means *until* ϕ_c becomes true, ϕ_a is true *implies* that *eventually* ϕ_b is true.

We denote $s_t \in S$ to be the MDP 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., $s_{t:t+k} = s_t s_{t+1} ... s_{t+k}$. scTLTL provides a set of real-valued functions that quantify the degree of satisfaction of a given $s_{0:T}$ with respective to a formula ϕ. This measure is also referred to as robustness degree or simply robustness ($\rho(s_{0:T}, \phi)$ maps a state trajectory and a formula to a real number). For example, $\rho(s_{0:3}, \Diamond(s < 4)) = \max(4 - s_0, 4 -$

$s_1, 4 - s_2)$. Here, if $4 - s_t > 0$, then $s_t < 4$ is satisfied. Because $\Diamond(s < 4)$ requires $s < 4$ to be true at least once in the trajectory, hence we take the max over the time horizon. In general, a robustness of greater than zero implies that $s_{t:t+k}$ satisfies ϕ and vice versa.

Definition 2. *An FSA corresponding to a scTLTL formula ϕ. is defined as a tuple $\mathcal{A}_\phi = \langle \mathbb{Q}_\phi, \Psi_\phi, q_{\phi,0}, p_\phi(\cdot|\cdot), \mathcal{F}_\phi \rangle$, where $\mathbb{Q}_\phi$ is a set of automaton states; Ψ_ϕ is the input alphabet (a set of first order logic formula); $q_{\phi,0} \in \mathbb{Q}_\phi$ is the initial state; $p_\phi : \mathbb{Q}_\phi \times \mathbb{Q}_\phi \to [0, 1]$ is a conditional probability defined as*

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

$\mathcal{F}_\phi$ is a set of final automaton states.

Here $q_{\phi,i}$ is the i^{th} automaton state of $\mathcal{A}_\phi$. $\psi_{q_{\phi,i},q_{\phi,j}} \in \Psi_\phi$ is the predicate guarding the transition from $q_{\phi,i}$ to $q_{\phi,j}$. Because $\psi_{q_{\phi,i},q_{\phi,j}}$ is a predicate without temporal operators, the robustness $\rho(s_{t:t+k}, \psi_{q_{\phi,i},q_{\phi,j}})$ is only evaluated at s_t. Therefore, we use the shorthand $\rho(s_t, \psi_{q_{\phi,i},q_{\phi,j}}) = \rho(s_{t:t+k}, \psi_{q_{\phi,i},q_{\phi,j}})$. The FSA corresponding to a scTLTL formula can be generated automatically with available packages like Lomap [30] (refer to [24] for details on the generation procedure).

IV. Problem Formulation

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, 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 using $\mathcal{M}$ and $\mathcal{A}_\phi$ and learn policy π_ϕ using the product.

Problem 2. *Given two scTLTL formula ϕ_1 and ϕ_2 and their optimal Q-functions $Q^\star_{\phi_1}$ and $Q^\star_{\phi_2}$, obtain the optimal policy $\pi^\star_{\phi_{\phi_1 \wedge \phi_2}}$ that satisfies $\phi_1 \wedge \phi_2$ and $\pi^\star_{\phi_{\phi_1 \vee \phi_2}}$ that satisfies $\phi_1 \vee \phi_2$.*

Here $Q^\star_{\phi_1}$ and $Q^\star_{\phi_2}$ can be the optimal Q-functions for the entropy-regularized MDP or the standard MDP. Problem 2 defines the problem of skill composition: given two policies each satisfying an scTLTL specification, construct the policy that satisfies the conjunction ($-AND-$)/disjunction ($-OR-$) of the given specifications. Solving this problem is useful when we want to break a complex task into simple and manageable components, learn a policy that satisfies each component and "stitch" all the components together so that the original task is satisfied. It can also be the case that as the scope of the task grows with time, the original task specification is amended with new items. Instead of having to re-learn the task from scratch, we can learn only policies that satisfies the new items and combine them with the old policy.

V. FSA Augmented MDP

In this section, we provide a solution to Problem 1 by constructing an augmented MDP using an FSA (generated from an scTLTL formula) and the original 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 outgoing transitions that originate from q (Ω_q is the set of automata states that are connected with q through outgoing edges).

Note that in Equation (7), $\tilde{r}$ is calculated from (s', q). It is a measure of the progress of satisfying D^q_ϕ by taking action a in state s (encapsulated by s').

We reduce Problem 1 to finding the optimal policy that maximizes the expected sum of discounted $\tilde{r}$, 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.

Intuitively, 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 which satisfies the TL specification (property of FSA) and hence Equation (5). Note that there can be infinitely many policies that satisfy Equation (5), $\pi^\star_\phi$ calculated from Equation (8) will be a subset of that from Equation (5) that put more value on policies that satisfy the formula with less number of steps.

The FSA augmented MDP can be constructed with any standard MDP and an 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. Automata Guided Skill Composition

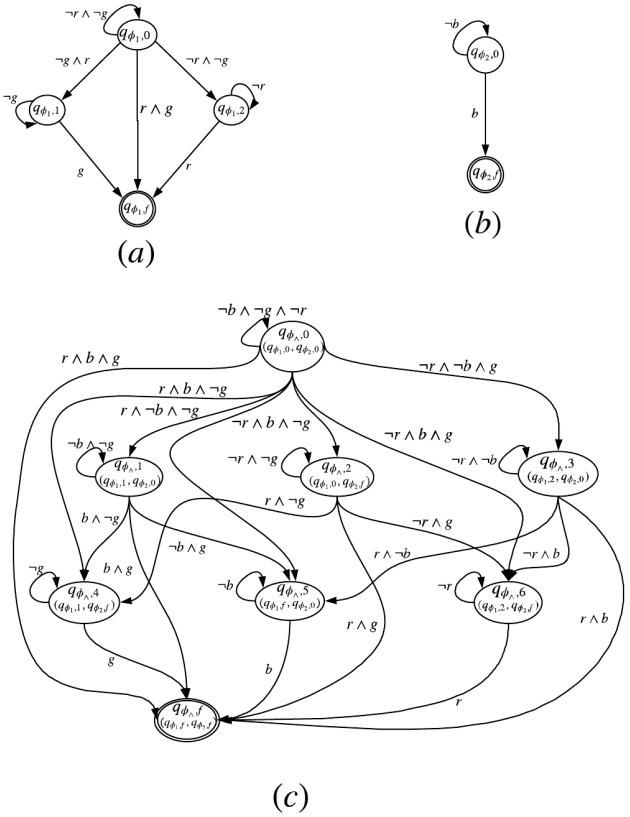

Fig. 1 : FSA for **(a)** $\phi_1 = \Diamond r \wedge \Diamond g$. **(b)** $\phi_2 = \Diamond b$. **(c)** $\phi_\wedge = \phi_1 \wedge \phi_2$.

In this section, we provide a solution for Problem 2 by constructing the FSA of $\phi_{\phi_1 \wedge \phi_2}$ from that of $\mathbb{A}_{\phi_1}$ and $\mathbb{A}_{\phi_2}$. Drawing inspiration from the synchronous product automaton [19], we introduce the following definition.

Definition 4. *Given* $\mathcal{A}_{\phi_1} = \langle \mathbb{Q}_{\phi_1}, \Psi_{\phi_1}, q_{\phi_1,0}, p_{\phi_1}, \mathcal{F}_{\phi_1} \rangle$ *and* $\mathcal{A}_{\phi_2} = \langle \mathbb{Q}_{\phi_2}, \Psi_{\phi_2}, q_{\phi_2,0}, p_{\phi_2}, \mathcal{F}_{\phi_2} \rangle$ *corresponding to formulas* ϕ_1 *and* ϕ_2*, the FSA of* $\phi_1 \wedge \phi_2$ *is the product automaton of* $\mathcal{A}_{\phi_1}$ *and* $\mathcal{A}_{\phi_1}$*, i.e.* $\mathcal{A}_{\phi_1 \wedge \phi_2} = \mathcal{A}_{\phi_1} \times \mathcal{A}_{\phi_2} = \langle \mathbb{Q}_{\phi_1 \wedge \phi_2}, \Psi_{\phi_1 \wedge \phi_2}, q_{\phi_1 \wedge \phi_2,0}, p_{\phi_1 \wedge \phi_2}, \mathcal{F}_{\phi_1 \wedge \phi_2} \rangle$ *where* $\mathbb{Q}_{\phi_1 \wedge \phi_2} = \mathbb{Q}_{\phi_1} \times \mathbb{Q}_{\phi_2}$ *is the set of product automaton states,* $q_{\phi_1 \wedge \phi_2,0} = (q_{\phi_1,0}, q_{\phi_2,0})$ *is the product initial state,* $\mathcal{F}_{\phi_1 \wedge \phi_2} = \mathcal{F}_{\phi_1} \cap \mathcal{F}_{\phi_2}$ *are the final accepting states. Following Definition 2, for states* $q_{\phi_1 \wedge \phi_2} = (q_{\phi_1}, q_{\phi_2}) \in \mathbb{Q}_{\phi_1 \wedge \phi_2}$ *and* $q'_{\phi_1 \wedge \phi_2} = (q'_{\phi_1}, q'_{\phi_2}) \in \mathbb{Q}_{\phi_1 \wedge \phi_2}$*, the transition probability* $p_{\phi_1 \wedge \phi_2}(q'_{\phi_1 \wedge \phi_2} | q_{\phi_1 \wedge \phi_2})$ *(written as* p *due to space constraints) is defined as*

$$p = \begin{cases} 1 & p_{\phi_1}(q'_{\phi_1}|q_{\phi_1}) p_{\phi_2}(q'_{\phi_2}|q_{\phi_2}) = 1 \\ 0 & \textit{otherwise}. \end{cases} \tag{9}$$

Example 1. *Figure 1 illustrates the FSA of* $\mathcal{A}_{\phi_1}$ *and* $\mathcal{A}_{\phi_2}$ *and their product automaton* $\mathcal{A}_{\phi_1 \wedge \phi_2}$*. Here* $\phi_1 = \Diamond r \wedge \Diamond g$ *which entails that both* r *and* g *needs to be true at least once (order does not matter), and* $\phi_2 = \Diamond b$ *means that* b *eventually needs to be true. The resultant product corresponds to the formula* $\phi = (\Diamond r \wedge \Diamond g) \wedge \Diamond b$*.* $(q_{\phi_1,f}, q_{\phi_2,f})$ *is to be reached to satisfy* $\phi_1 \wedge \phi_2$*. However, if* $\phi_1 \vee \phi_2$ *is the goal, then reaching any of the states with* $q_{\phi_i,f}, i \in \{1,2\}$ *satisfies the goal (*$F_{\phi_1 \vee \phi_2} = \mathcal{F}_{\phi_1} \cup \mathcal{F}_{\phi_2}$*).*

We provide the following theorem on automata guided skill composition:

Theorem 2. *Let* $\boldsymbol{Q}_\phi^{\pi^\star} = \{Q_{\phi_1}^{\pi_1^\star}, ..., Q_{\phi_n}^{\pi_n^\star}\}$ *be a set with entries* $Q_{\phi_i}^{\pi_i^\star}$ *being the optimal Q-function for the FSA augmented MDP* $\mathcal{M}_{\phi_i}$*. The optimal Q-function for* $\mathcal{M}_{\phi_\wedge}$ *where* $\phi_\wedge = \bigwedge_i \phi_i$ *is* $Q_{\phi_\wedge}^{\pi_\wedge^\star} = \max(\boldsymbol{Q}_\phi^{\pi^\star})$.

Proof: For $q_{\phi_\wedge} = (q_{\phi_1}, q_{\phi_2}) \in \mathbb{Q}_{\phi_\wedge}$, let $\Psi_{q_{\phi_\wedge}}$, $\Psi_{q_{\phi_1}}$ and $\Psi_{q_{\phi_2}}$ denote the set of predicates guarding the edges originating from $q_{\phi_\wedge}$, q_{ϕ_1} and q_{ϕ_2} respectively. Equation (9) entails that a transition at $q_{\phi_\wedge}$ in the product automaton $\mathcal{A}_{\phi_\wedge}$ exists only if corresponding transitions at q_{ϕ_1}, q_{ϕ_2} exist in $\mathcal{A}_{\phi_1}$ and $\mathcal{A}_{\phi_2}$ respectively. Therefore, $\psi_{q_{\phi_\wedge}, q'_{\phi_\wedge}} = \psi_{q_{\phi_1}, q'_{\phi_1}} \wedge \psi_{q_{\phi_2}, q'_{\phi_2}}$, for $\psi_{q_{\phi_\wedge}, q'_{\phi_\wedge}} \in \Psi_{q_{\phi_\wedge}}, \psi_{q_{\phi_1}, q'_{\phi_1}} \in \Psi_{q_{\phi_1}}, \psi_{q_{\phi_2}, q'_{\phi_2}} \in \Psi_{q_{\phi_2}}$ (here q'_{ϕ_i} is a state such that $p_{\phi_i}(q'_{\phi_i}|q_{\phi_i}) = 1$). Therefore, we have

$$D_{\phi_\wedge}^{q_{\phi_\wedge}} = \bigvee_{q'_{\phi_1}, q'_{\phi_2}} (\psi_{q_{\phi_1}, q'_{\phi_1}} \wedge \psi_{q_{\phi_2}, q'_{\phi_2}}) \tag{10}$$

where q'_{ϕ_1}, q'_{ϕ_2} don't equal to q_{ϕ_1}, q_{ϕ_2} at the same time (to avoid self looping edges). Using the fact that $\psi_{q_{\phi_i}, q_{\phi_i}} = \neg \bigvee_{q'_{\phi_i} \neq q_{\phi_i}} \psi_{q_{\phi_i}, q'_{\phi_i}}$ and repeatedly applying the distributive laws $(\Delta \wedge \Omega_1) \vee (\Delta \wedge \Omega_2) = \Delta \wedge (\Omega_1 \vee \Omega_2)$ and $(\Delta \vee \Omega_1) \wedge (\Delta \vee \Omega_2) = \Delta \vee (\Omega_1 \wedge \Omega_2)$ to $D_{\phi_\wedge}^{q_{\phi_\wedge}}$, we arrive at

$$\begin{aligned} D_{\phi_\wedge}^{q_{\phi_\wedge}} &= \Big(\bigvee_{q'_{\phi_1} \neq q_{\phi_1}} \psi_{q_{\phi_1}, q'_{\phi_1}} \Big) \vee \Big(\bigvee_{q'_{\phi_2} \neq q_{\phi_2}} \psi_{q_{\phi_2}, q'_{\phi_2}} \Big) \\ &= D_{\phi_1}^{q_{\phi_1}} \vee D_{\phi_2}^{q_{\phi_2}}. \end{aligned} \tag{11}$$

Let $\tilde{r}_{\phi_\wedge}, \tilde{r}_{\phi_1}, \tilde{r}_{\phi_2}$ and $\tilde{s}_{\phi_\wedge}, \tilde{s}_{\phi_1}, \tilde{s}_{\phi_2}$ be the reward functions and states for the FSA augmented MDPs $\mathcal{M}_{\phi_\wedge}, \mathcal{M}_{\phi_1}, \mathcal{M}_{\phi_2}$ respectively. $s_{\phi_\wedge}, s_{\phi_1}, s_{\phi_2}$ are the states for the corresponding MDPs. Plugging Equation (11) into Equation (7) and using the robustness definition for disjunction results in

$$\begin{aligned} \tilde{r}_{\phi_\wedge}(\tilde{s}_{\phi_\wedge}, \tilde{s}'_{\phi_\wedge}) &= \rho(s'_{\phi_\wedge}, D_{\phi_\wedge}^{q_{\phi_\wedge}}) \\ &= \rho(s'_{\phi_\wedge}, D_{\phi_1}^{q_{\phi_1}} \vee D_{\phi_2}^{q_{\phi_2}}) \\ &= \max\big(\rho(s'_{\phi_1}, D_{\phi_1}^{q_{\phi_1}}), \rho(s'_{\phi_2}, D_{\phi_2}^{q_{\phi_2}})\big) \\ &= \max\big(\tilde{r}_{\phi_1}(\tilde{s}_{\phi_1}, \tilde{s}'_{\phi_1}), \tilde{r}_{\phi_2}(\tilde{s}_{\phi_2}, \tilde{s}'_{\phi_2})\big). \end{aligned} \tag{12}$$

Looking at Theorem 1, the log-sum-exponential of the composite reward $\mathrm{r} = \alpha \log(|| \exp(\boldsymbol{r}/\alpha)||_{\boldsymbol{w}})$ is an approximation of the maximum function. In the low temperature limit we have $r \to \max(\boldsymbol{r})$ as $\alpha \to 0$. Applying Corollary 1 results in Theorem 2. ∎

Similar to Corollary 1, Theorem 2 does not require entropy-regularized RL and can be applied to any actor-critic methods.

Having obtained the optimal Q-function, a policy can be constructed by taking the greedy step with respective to the Q-function in the discrete action case. For the case of continuous action space where the policy is represented by a function approximator, the policy update procedure in actor-critic methods can be used to extract a policy from the Q-function.

As is mentioned in Example 1, $-AND-$ and $-OR-$ task compositions follow the same procedure in our framework (Theorem 2). The only difference is the termination condition. For $-AND-$ tasks, the final set of states $\mathcal{F}_{\phi_\wedge} = \bigcap \mathcal{F}_{\phi_i}$ (i.e. all the constituent FSAs are required to reach their final states). Whereas for $-OR-$ tasks $\mathcal{F}_{\phi_\vee} = \bigcup \mathcal{F}_{\phi_i}$. A summary of the composition procedure is provided in Algorithm 1.

Algorithm 1 Automata Guided Skill Composition

1: **Inputs**: scTLTL task specification ϕ_1 and ϕ_2, randomly initialized policies π_{ϕ_1}, π_{ϕ_2} and action-value functions $Q^{\pi_{\phi_1}}_{\phi_1}$, $Q^{\pi_{\phi_2}}_{\phi_2}$. State and action spaces of the MDP.
2: Construct FSA augmented MDPs $\mathcal{M}_{\phi_1}$ and $\mathcal{M}_{\phi_2}$ ▷ using Definition 3
3: $\pi^\star_{\phi_1}, Q^{\pi^\star_{\phi_1}}_{\phi_1}, \mathcal{B}_{\phi_1} \leftarrow ActorCritic(\mathcal{M}_{\phi_1})$ ▷ learns the optimal policy and Q-function
4: $\pi^\star_{\phi_2}, Q^{\pi^\star_{\phi_2}}_{\phi_2}, \mathcal{B}_{\phi_2} \leftarrow ActorCritic(\mathcal{M}_{\phi_2})$
5: $Q^{\pi^\star_{\phi_2\wedge\phi_2}}_{\phi_2\wedge\phi_2} = \max(Q^{\pi^\star_{\phi_1}}_{\phi_1}, Q^{\pi^\star_{\phi_2}}_{\phi_2})$ ▷ construct the optimal composed Q-function using Theorem 2
6: $\mathcal{B}_{\phi_\wedge} \leftarrow ConstructProductBuffer(\mathcal{B}_{\phi_1}, \mathcal{B}_{\phi_2})$
7: $\pi^\star_{\phi_2\wedge\phi_2} \leftarrow ExtractPolicy(Q^{\pi^\star_{\phi_2\wedge\phi_2}}_{\phi_2\wedge\phi_2}, \mathcal{B}_{\phi_\wedge})$
8: **return** $\pi^\star_{\phi_2\wedge\phi_2}$

In Algorithm 1, steps 3 and 4 seeks to obtain the optimal policies and Q-functions using an off-policy actor-critic algorithm. $\mathcal{B}_{\phi_1}$ and $\mathcal{B}_{\phi_2}$ are the replay buffers collected while training for each skill. Step 6 constructs the product replay buffer for policy extraction. This step is necessary because each $\mathcal{B}_{\phi_i}$ contains state of form (s, q_i), $i \in \{1, 2\}$ whereas the composed policy takes state (s, q_1, q_2) as input (Definition 4). Therefore, we transform each experience $((s, q_i), a, (s', q'_i), r)$ to $((s, q_i, q_{j\neq i}), a, (s', q'_i, q'_{j\neq i}), r)$ where $q_{j\neq i}$ is chosen at random from the automaton states of $\mathcal{A}_{\phi_j}$ and $q'_{j\neq i}$ is calculated using Equation (6). The reward r will not be used in policy extraction as the composed Q-function will not be updated. Step 7 extracts the optimal composed policy from the optimal composed Q-function (this corresponds to running only the policy update step in the actor critic algorithm).

VII. Case Studies

In this section, We evaluate our automata guided learning and composition methods. The first environment is a simple 2D grid world environment that is used for proof of concept and policy visualization. The second environment is a robot manipulation environment. In all experiments, we use Lomap [30] to automatically translate an scTLTL specification to its corresponding FSA.

A. Grid World

Consider an agent that navigates in a 8×10 grid world as shown in Figure 3 . Its MDP states (x, y) are the agent's integer coordinates on the grid. The agent's actions include $A = \{$*up*, *down*, *left*, *right*, *stay*$\}$. The transitions are such that for each action command, the agent follows that command with probability 0.8 or chooses a random action with probability 0.2. We train the agent on two tasks, $\phi_1 = \Diamond r \wedge \Diamond g$ and $\phi_2 = \Diamond b$ (same as in Example 1). The regions are defined by the predicates $r = (1 < x < 3) \wedge (1 < y < 3)$, $g = (4 < x < 6) \wedge (4 < y < 6)$ and $b = (1 < x < 3) \wedge (6 < y < 8)$. Because the coordinates are integers, a and b define a point goal rather than regions.

We apply standard tabular Q-learning [31] on the FSA augmented MDP of this environment. For all experiments, we use a discount factor of 0.95, learning rate of 0.1, episode horizon of 200 steps, a random exploration policy and a total number of 2000 update steps which is enough to reach convergence (learning curve is neglected).

Figure 3 (a) and (b) show the learned optimal policies obtained using the FSA augmented MDP in the form of sub-policies extracted using $\pi^\star_{\phi_i}(x, y, q_{\phi_i}) = \arg\max_a Q^\star_{\phi_i}(x, y, q_{\phi_i}, a)$. We can observed that each $\pi^\star_{\phi_i}(x, y, q_{\phi_i})$ has goals given by Equation (7). The agent starts at $q_{\phi_i,0}, i \in \{1, 2\}$, by reaching these goals and transitioning on the FSA, the specification is eventually satisfied.

Figure 3 (c) shows the composed policy for task $\phi_\wedge = \phi_1 \wedge \phi_2$ obtained using Theorem 2. It is clear that the composed policy is able to act optimally in terms of maximizing the expected sum of discounted reward given by Equation (12). Following the composed policy and transitioning on the FSA in Figure 1 (c) leads to satisfaction of $\phi_\wedge$ $(-AND-)$. As discussed in the previous section, if the $-OR-$ task is desired, following the same composed policy and terminating at any of the states $q_{\phi_\wedge,2}$, $q_{\phi_\wedge,4}$, $q_{\phi_\wedge,5}$, $q_{\phi_\wedge,6}$, $q_{\phi_\wedge,f}$ will satisfy $\phi_\vee = \phi_1 \vee \phi_2$.

B. Robotic Manipulation

1) Experiment Setup: Figure 2 shows our experiment setup. Our policy controls the 7 degree-of-freedom joint velocities of the right arm of a Baxter robot. In front of the robot there are three circular regions (red, green, blue plates) and it has to learn to traverse the regions in user specified ways. The positions of the plates are tracked using a motion capture system and thus fully observable. In addition, we also track the position of one of the user's hands (by wearing a glove with trackers attached). Our MDP state space is 22 dimensional that includes 7 joint angles, xyz positions of the three regions (denoted by $\boldsymbol{p}^r$, $\boldsymbol{p}^g$, $\boldsymbol{p}^b$), the user's hand ($\boldsymbol{p}^h$) and the robot's end-effector ($\boldsymbol{p}^{ee}$). The state and action spaces are continuous in this case. We train in simulation using the V-REP simulator (Figure 2 (a)) [23] and evaluate the policies on the real robot.

We define the following predicates

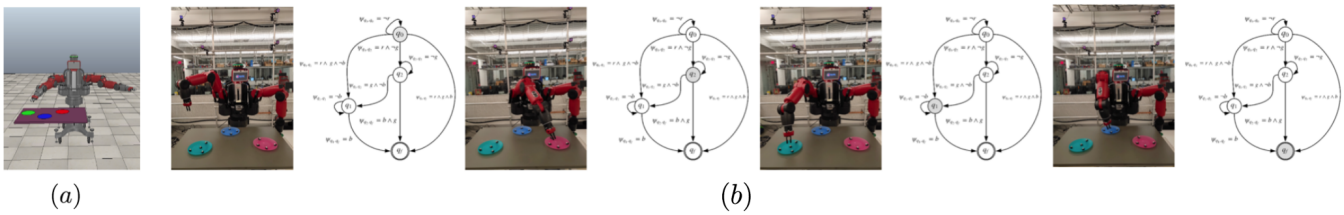

Fig. 2 : **(a)** Simulation setup. **(b)** Sample execution of $\phi_{traverse}$ with FSA transitions shown. The shaded q state represents the current automaton state (the FSAs here are the same as in Figure 4 (a), they are included to demonstrate the transitions).

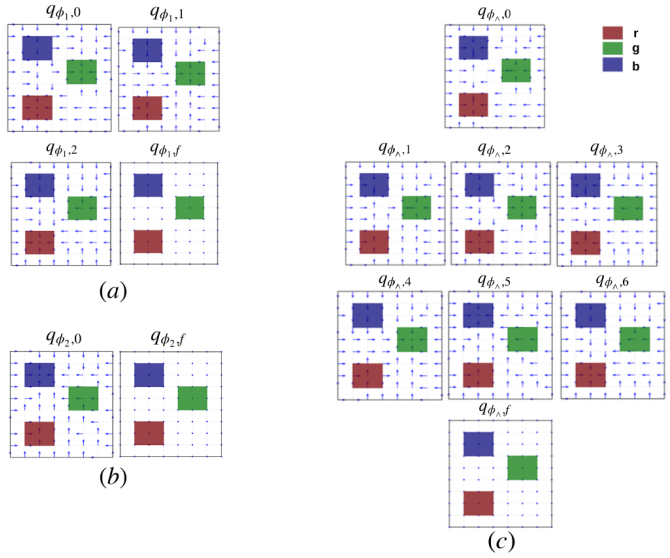

Fig. 3 : Policies for **(a)** $\phi_1 = \Diamond r \wedge \Diamond g$. **(b)** $\phi_2 = \Diamond b$. **(c)** $\phi_\wedge = \phi_1 \wedge \phi_2$. The agent moves in a 8×10 gridworld with 3 labeled regions. The agent has actions $\{$*up*, *down*, *left*, *right*, *stay*$\}$ where the directional actions are represented by arrows, *stay* is represented by the blue dot.

1) $\psi_i = ||\boldsymbol{p}^i - \boldsymbol{p}^{ee}|| < \epsilon$, $i \in \{r, g, b, h\}$ where r, g, b denotes the color of the regions. h represents the user's hand. ϵ is a threshold which we set to be 5 centimeters. ψ_i constrains the relative distance between the robot's end-effector and the selected object.
2) $\psi_{hand\_in\_sight} = (x_{min} < \boldsymbol{p}^h_x < x_{max}) \wedge (y_{min} < \boldsymbol{p}^h_y < y_{max}) \wedge (z_{min} < \boldsymbol{p}^h_z < z_{max})$. This predicate evaluates to true if the user's hand appears in the cubic region defined by $[x_{min}, x_{max}, y_{min}, y_{max}, z_{min}, z_{max}]$.

We test our method on the following composition task

1) $\phi_{traverse} = \Diamond(\psi_r \wedge \Diamond(\psi_g \wedge \Diamond\psi_b))$
Description: traverse the three regions in the order of red, green and blue.
2) $\phi_{interrupt} = (\psi_{hand\_in\_sight} \Rightarrow \Diamond\psi_h)\ \mathcal{U}\ \psi_b$
Description: before reaching the blue region, if the user's hand appears in sight, then eventually reach for the user's hand, otherwise just reach for the blue region.
3) $\phi_\wedge = \phi_{traverse} \wedge \phi_{interrupt}$
Description: conjunction of the first two tasks.

The FSAs for $\phi_{traverse}$ and $\phi_{interrupt}$ are presented in Figure 4 . The FSA for $\phi_\wedge$ (14 nodes and 72 edges) is not presented here due to space constraints.

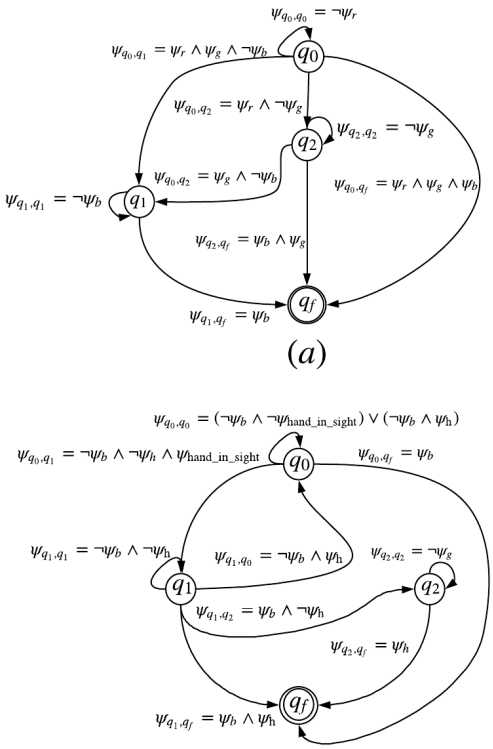

Fig. 4 : The FSA for **(a)**:$\phi_{traverse} = \Diamond(\psi_r \wedge \Diamond(\psi_g \wedge \Diamond\psi_b))$ **(b)** $\phi_{interrupt} = (\psi_{hand\_in\_sight} \Rightarrow \Diamond\psi_h)\ \mathcal{U}\ \psi_b$.

2) Implementation Details: The policies and Q-functions for all tasks in this section are represented by a feed-forward neural network (3 layers with 300, 200, 100 ReLU units respectively). For tasks $\phi_{traverse}$ and $\phi_{interrupt}$, the input state space is 23 dimensional (22 continuous dimensional MDP state and 1 discrete dimension for the automaton state). For task $\phi_\wedge$, the state space is 24 dimensional (2 discrete dimensions for the product automaton state).

We use soft actor-critic (sac) [11] and deep deterministic policy gradient [16] to train the FSA augmented MDP for each of the two tasks. Implementation of the soft actor-critic algorithm follows [28]. Algorithm 1 is used to obtain the policy for $\phi_\wedge$. After each episode, the joint angles, the FSA state, the position of the plates as well as the position of the hand are randomly reinitialized (within certain boundaries) to

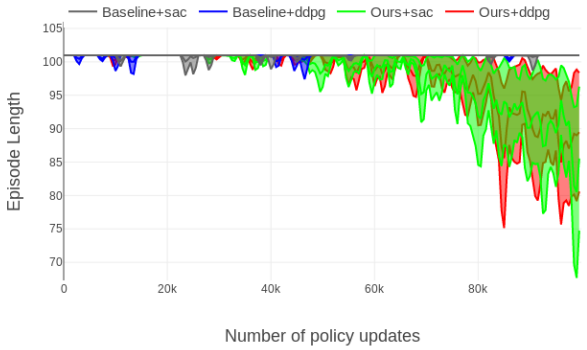

Fig. 5 : Average episode length as a function of policy update steps for task $\phi_{traverse}$ (smaller number means faster completion of the task). The mean and standard deviation are calculated from 5 episodes. Training is performed in simulation.

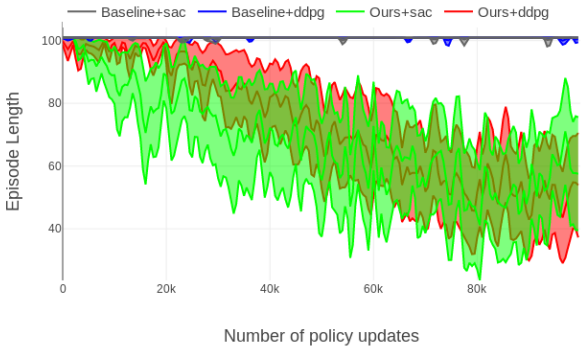

Fig. 6 : Average episode length as a function of policy update steps for task $\phi_{interrupt}$

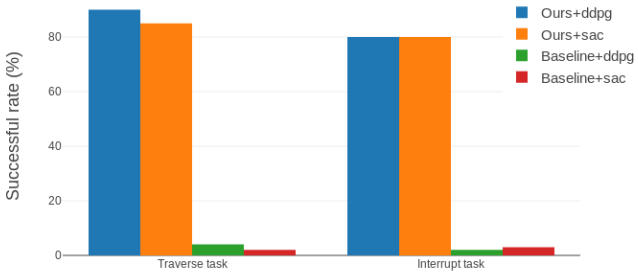

Fig. 7 : Evaluation success rate on the real robot over 20 trials for tasks $\phi_{traverse}$ and $\phi_{interrupt}$.

ensure generalization across different task configurations. The robot is controlled at 20 Hz. Each episode is 100 time-steps (about 5 seconds). The episode restarts if the final automaton state q_f is reached. During training, we perform 100 policy and Q-function updates with a batch of 5 trajectories. All of our training is performed in simulation and for this set of tasks, the policy is able to transfer to the real robot without further fine-tuning.

3) Comparison Cases: To evaluate our method and the effectiveness of the reward in Equation (7), we compare with a baseline reward of the form $\tilde{r}_b(\tilde{s}, \tilde{s}') = \mathbb{1}(q' \neq q) - \delta$, $(0 < \delta < 1)$ where $\mathbb{1}()$ is the indicator function. In the comparison case, the agent receives a reward of $1-\delta$ if a transition is made in the FSA, else it obtains a small negative reward $-\delta$ for each step taken (we use $\delta = 0.2$). We acknowledge that with enough effort, an efficient reward for any task can be manually designed. Our goal here is to automatically generate dense rewards given high level specifications. Therefore, we do not put much effort in designing a reward for the comparison case.

To evaluate the automata guided skill composition, we compare the composed policy for task $\phi_\wedge$ with a policy for the same task learned from scratch. We also compare our composition method with the average Q-function composition proposed in [10].

The method we propose is a modification on the MDP that can be trained with existing RL algorithms. Therefore, we try to keep the RL algorithm fixed for fair comparisons.

4) Results and Discussion: Learning curves are presented in Figure 5 and Figure 6 . The episode length is used as the evaluation metric because we are comparing 2 different reward formulations (there is a scale difference in the discounted return). An episode is terminated either when the horizon is reached (100 steps) or the final state in the FSA is reached. Therefore, a shorter episode length indicates faster completion of the task. In the figures, we denote training with reward $\tilde{r}$ in Equation (7) as *Ours*. Training with reward $\tilde{r}_b$ is referred to as *Baseline*.

We can observe from the learning curves that our reward formulation is able to eventually complete both tasks with consistency whereas using the baseline reward fails to learn either task. This is largely due to the fact that the robustness degree used to formulate $\tilde{r}$ provides a continuous measure of satisfaction which is a dense reward signal. In general, $\phi_{interrupt}$ is easier to satisfy than $\phi_{traverse}$ as the former takes less FSA transitions to reach the final state. This is also reflected in the results. Figure 2 (b) shows an execution trace of task $\phi_{traverse}$ with the corresponding transitions on the FSA. Figure 6 shows the success rate of 20 evaluation trials for both tasks on the real robot.

Figure 9 shows the learning curves for the composed task $\phi_\wedge$. In the figure, *ddpg* and *sac* denote learning $\phi_\wedge$ from scratch using the FSA augmented MDP. *max* represents task composition using Algorithm 1 (we use the maximum of the Q-functions for composition) and *average* represents task composition using the average of the Q-functions [9]. *sac+average* means average Q-composition using policies learned from soft actor-critic. Composition is limited to 40K updates whereas learning from scratch is allowed 100K updates.

We can observe that given the allowed training time, nor

Fig. 8 : An execution trace of of the composed policy for task $\phi_\wedge = \phi_{traverse} \wedge \phi_{interrupt}$.

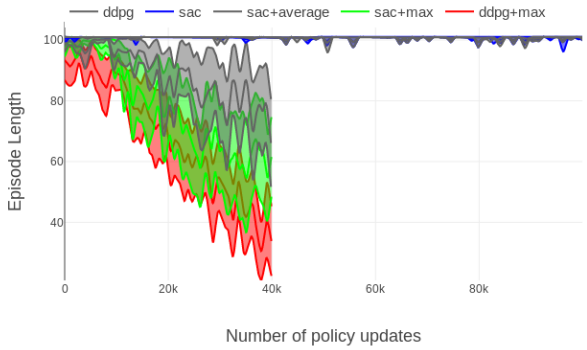

Fig. 9 : Learning/composition curve for task $\phi_\wedge$. Here *ddpg* and *sac* denote learning of $\phi_\wedge$ from scratch using the FSA augmented MDP. *max* indicates task composition using Algorithm 1. *average* indicates task composition using the average of the Q-functions [9].

ddpg or sac is able to learn the task from scratch. This is likely due to the large size of the FSA (14 nodes and 72 edges). However, we are able to obtain satisfying policies for all composition case. Within our experiments, the automata guided skill composition with ddpg (*ddpg+max*) results in fastest learning and lowest variance compared to the others. *sac+max* is able to perform comparatively with slightly higher variance (due to the stochastic nature of the policy). Composition using the averaged Q-functions is also able to obtain the skill but requires more steps to complete the task. This set of results show that our framework is compatible with different composition methods while imposing no constraints on the policy structure. Figure 10 shows the evaluation success rates on the real robot.

Figure 8 shows an execution trace for task $\phi_\wedge$ using the composed policy. Notice that when the robot detects that the user's hand appears in the region defined by $\psi_{hand\_in\_sight}$, it stops the traverse task to satisfy the interrupt task. After it reaches for the user's hand, it goes on to satisfy the rest of the specification.

It is important to note that the number of policy updates does not translate directly to training wall-time. Because composition is performed on collected data from training the constituent policies, no further exploration is necessary (which is the time bottleneck in our case). In our experiments, exploration and data collection take about three quarter of the entire training time (around 3 hours).

We would like to point out that even though Figure 9 suggests that learning $\phi_\wedge$ from scratch failed in the allocated time, the agents are in fact making progress in terms of increasing discounted return. Running these experiments for 200K update steps result in policies with around 40% success rate for task $\phi_\wedge$.

A potential drawback of our composition method (or any such methods based on combining Q-functions) is scale sensitivity. In our experiments, the reward used to train $\phi_{traverse}$ and $\phi_{interrupt}$ is of the same scale. If this condition does not hold (one task has rewards orders of magnitude larger than the other), simply combining Q-functions will not result in the desired policy. We also face the problem of dimensional explosion of automaton states when composing many policies. These are interesting problems to look at in the future.

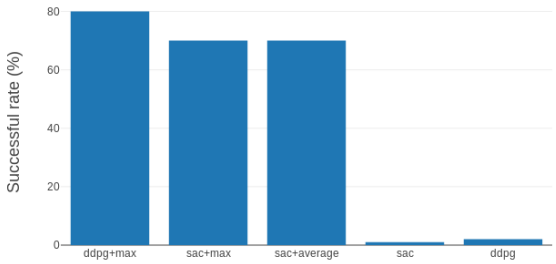

Fig. 10 : Evaluation success rates on the real robot over 20 trials for the composition task $\phi_\wedge$.

VIII. Conclusion

In this paper, we present a framework that supports learning of tasks specified as an scTLTL formula. This includes task structures such as temporal sequencing, choices, ambiguity and combinations of them. We also introduce a method to compose policies learned with our framework to obtain new skills without additional exploration. We have shown the applicability of our techniques in learning and composing robotic manipulation policies. Future work includes extending this framework to support full TLTL specifications over states and actions. We also plan to apply this framework to learning of more complex tasks to fully explore its potential.

References

[1] Derya Aksaray, Austin Jones, Zhaodan Kong, Mac Schwager, and Calin Belta. Q-learning for robust satisfaction of signal temporal logic specifications. In Decision and Control (CDC), 2016 IEEE 55th Conference on, pages 6565–6570. IEEE, 2016.

[2] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In AAAI, 2018.

[3] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In ICML, 2017.

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

[5] Marco Da Silva, Frédo Durand, and Jovan Popović. Linear bellman combination for control of character animation. Acm transactions on graphics (tog), 28(3): 82, 2009.

[6] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In ICML, 2017.

[7] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. Reinforcement learning for ltlf/ldlf goals. CoRR, abs/1807.06333, 2018.

[8] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In ICML, 2017.

[9] Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. Composable deep reinforcement learning for robotic manipulation. CoRR, abs/1803.06773, 2018.

[10] Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. Composable deep reinforcement learning for robotic manipulation. arXiv preprint arXiv:1803.06773, 2018.

[11] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. CoRR, abs/1812.05905, 2018.

[12] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In International Conference on Machine Learning, pages 2112–2121, 2018.

[13] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. CoRR, abs/1806.10293, 2018.

[14] Tejas D. Kulkarni, Karthik R. Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. preprint arXiv:1604.06057, 2016.

[15] Xiao Li, Cristian-Ioan Vasile, and Calin Belta. Reinforcement learning with temporal logic rewards. arXiv preprint arXiv:1612.03471, 2016.

[16] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. CoRR, abs/1509.02971, 2015.

[17] Jeffrey Mahler, Matthew Matl, Xinyu Liu, Albert Li, David V. Gealy, and Kenneth Y. Goldberg. Dex-net 3.0: Computing robust robot suction grasp targets in point clouds using a new analytic model and deep learning. CoRR, abs/1709.06670, 2017.

[18] Andrychowicz Marcin, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. arXiv preprint arXiv:1808.00177, 2018.

[19] Vince Molnár and András Vörös. Synchronous product automaton generation for controller optimization.

[20] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In ICML, 1999.

[21] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: example-guided deep reinforcement learning of physics-based character skills. ACM Trans. Graph., 37:143:1–143:14, 2018.

[22] Ivaylo Popov, Nicolas Heess, Timothy P. Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin A. Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. CoRR, abs/1704.03073, 2017.

[23] Eric Rohmer, Surya P. N. Singh, and Marc Freese. V-rep: A versatile and scalable robot simulation framework. 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 1321–1326, 2013.

[24] Kristin Yvonne Rozier. Explicit or symbolic translation of linear temporal logic to automata. PhD thesis, Rice University, 2013.

[25] John Schulman, Pieter Abbeel, and Xi Chen. Equivalence between policy gradients and soft q-learning. CoRR, abs/1704.06440, 2017.

[26] Tianmin Shu, Caiming Xiong, and Richard Socher. Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. CoRR, abs/1712.07294, 2017.

[27] Emanuel Todorov. Compositionality of optimal control laws. In Advances in Neural Information Processing Systems, pages 1856–1864, 2009.

[28] Kristian Hartikainen George Tucker Sehoon Ha Jie Tan

Vikash Kumar Henry Zhu Abhishek Gupta Pieter Abbeel Tuomas Haarnoja, Aurick Zhou and Sergey Levine. Soft actor-critic algorithms and applications. Technical report, 2018.

[29] Benjamin van Niekerk, Steven James, Adam Christopher Earle, and Benjamin Rosman. Will it blend? composing value functions in reinforcement learning. CoRR, abs/1807.04439, 2018.

[30] C Vasile. Github repository, 2017.

[31] Christopher John Cornish Hellaby Watkins. Learning From Delayed Rewards. PhD thesis, King's College, Cambridge, England, 1989.

[32] Min Wen, Ivan Papusha, and Ufuk Topcu. Learning from demonstrations with high-level side information. In IJCAI, 2017.

[33] Yuke Zhu, Daniel Gordon, Eric Kolve, Dieter Fox, Li Fei-Fei, Abhinav Gupta, Roozbeh Mottaghi, and Ali Farhadi. Visual semantic planning using deep successor representations. arXiv preprint ArXiv:1705.08080, pages 1–13, 2017.