The Logical Options Framework

Brandon Araki [1] **Xiao Li** [1] **Kiran Vodrahalli** [2] **Jonathan DeCastro** [3] **J. Micah Fry** [4] **Daniela Rus** [1]

Abstract

Learning composable policies for environments with complex rules and tasks is a challenging problem. We introduce a hierarchical reinforcement learning framework called the *Logical Options Framework* (LOF) that learns policies that are *satisfying*, *optimal*, and *composable*. LOF efficiently learns policies that satisfy tasks by representing the task as an automaton and integrating it into learning and planning. We provide and prove conditions under which LOF will learn satisfying, optimal policies. And lastly, we show how LOF's learned policies can be composed to satisfy unseen tasks with only 10-50 retraining steps on our benchmarks. We evaluate LOF on four tasks in discrete and continuous domains, including a 3D pick-and-place environment.

1. Introduction

To operate in the real world, intelligent agents must be able to make long-term plans by reasoning over symbolic abstractions while also maintaining the ability to react to low-level stimuli in their environment (Zhang & Sridharan, 2020). Many environments obey rules that can be represented as logical formulae; e.g., the rules a driver follows while driving, or a recipe a chef follows to cook a dish. Traditional motion and path planning techniques struggle to plan over these long-horizon tasks, but hierarchical approaches such as hierarchical reinforcement learning (HRL) can solve lengthy tasks by planning over both the high-level rules and the low-level environment. However, solving these problems involves trade-offs among multiple desirable properties, which we identify as *satisfaction*, *optimality*, and *composability* (described below). Today's hierarchical planning algorithms lack at least one of these objectives. For example, Reward Machines (Icarte et al., 2018) are satisfying and optimal, but not composable; the options framework (Sutton et al., 1999) is composable and hierarchically optimal, but cannot satisfy specifications. An algorithm that achieves all three of these properties would be very powerful because it would enable a model learned on one set of rules to generalize to arbitrary rules. We introduce the *Logical Options Framework*, which builds upon the options framework and aims to combine symbolic reasoning and low-level control to achieve satisfaction, optimality, and composability with as few compromises as possible. Furthermore, we demonstrate that models learned with our framework generalize to arbitrary sets of rules without any further learning, and we also show that our framework is compatible with arbitrary domains and planning algorithms, from discrete domains and value iteration to continuous domains and proximal policy optimization (PPO).

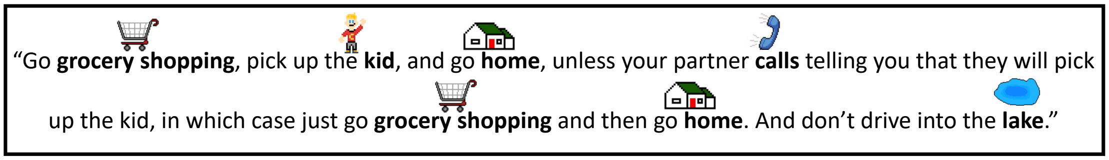
Satisfaction: An agent operating in an environment governed by rules must be able to satisfy the specified rules. Satisfaction is a concept from formal logic, in which the input to a logical formula causes the formula to evaluate to `True`. Logical formulae can encapsulate rules and tasks like the ones described in Fig. 1, such as "pick up the groceries" and "do not drive into a lake". In this paper, we state conditions under which our method is guaranteed to learn satisfying policies.

Optimality: Optimality requires that the agent maximize its expected cumulative reward for each episode. In general, satisfaction can be achieved by rewarding the agent for satisfying the rules of the environment. In hierarchical planning there are several types of optimality, including hierarchical optimality (optimal with respect to the hierarchy) and optimality (optimal with respect to everything). We prove in this paper that our method is hierarchically optimal and, under certain conditions, optimal.

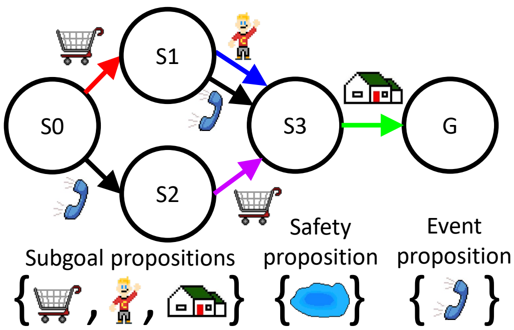
Composability: Our method is also composable – once it has learned the low-level components of a task, the learned model can be rearranged to satisfy arbitrary tasks. More specifically, the rules of an environment can be factored into liveness and safety properties, which we discuss in Sec. 3. The learned model has high-level actions called options that can be composed to satisfy new liveness properties. A shortcoming of many RL models is that they are not composable – trained to solve one specific task, they are incapable of han-

[1]CSAIL, Massachusetts Institute of Technology, Cambridge, MA, USA [2]Department of Computer Science, Columbia University, New York City, NY, USA [3]Toyota Research Institute, Cambridge, MA, USA [4]MIT Lincoln Laboratory, Lexington, MA, USA. Correspondence to: Brandon Araki <araki@csail.mit.edu>.

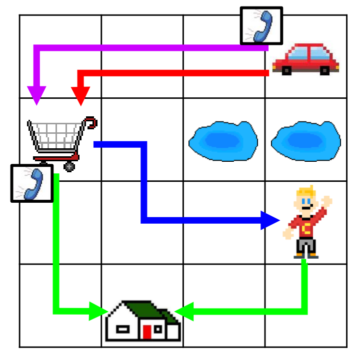
Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.



(a) These natural language instructions can be transformed into an FSA, shown in (b).



(b) The FSA representing the natural language instructions. The propositions are divided into "subgoal", "safety", and "event."



(c) The low-level MDP and corresponding policy that satisfies the instructions.

Figure 1. Many parents face this task after school ends – who picks up the kid, and who gets groceries? The pictorial symbols represent propositions, which are true or false depending on the state of the environment. The arrows in (c) represent sub-policies, and the colors of the arrows match the corresponding transition in the FSA. The boxed phone at the beginning of some of the arrows represents how these sub-policies can occur only after the agent receives a phone call.

dling even small variations in the task structure. However, the real world is a dynamic and unpredictable place, so the ability to use a learned model to automatically reason over as-yet-unseen tasks is a crucial element of intelligence.

Fig. 1 gives an example of how LOF works. The environment is a world with a grocery store, your (hypothetical) kid, your house, and some lakes, and in which you, the agent, are driving a car. The propositions are divided into "subgoals", representing events that can be achieved, such as going grocery shopping; "safety" propositions, representing events that you must avoid (driving into a lake); and "event" propositions, corresponding to events that you have no control over (receiving a phone call) (Fig. 1b). In this environment, you have to follow rules (Fig. 1a). These rules can be converted into a logical formula, and from there into a finite state automaton (FSA) (Fig. 1b). LOF learns an option for each subgoal (illustrated by the arrows in Fig. 1c), and a meta-policy for choosing amongst the options to reach the goal state of the FSA. After learning, the options can be recombined to fulfill arbitrary tasks.

1.1. Contributions

This paper introduces the Logical Options Framework (LOF) and makes four contributions to the hierarchical reinforcement learning literature:

1. The definition of a hierarchical semi-Markov Decision Process (SMDP) that is the product of a logical FSA and a low-level environment MDP.
2. A planning algorithm for learning options and meta-policies for the SMDP that allows the options to be composed to solve new tasks with only 10-50 retraining steps on our benchmarks and no additional samples from the environment.
3. Conditions and proofs for satisfaction and optimality.
4. Experiments on a discrete delivery domain, a continuous 2D reacher domain, and a continuous 3D pick-and-place domain on four tasks demonstrating satisfaction, optimality, and composability.

2. Background

Linear Temporal Logic: We use linear temporal logic (LTL) to formally specify rules (Clarke et al., 2001). LTL can express tasks and rules using temporal operators such as "eventually" and "always." LTL formulae are used only indirectly in LOF, as they are converted into automata that the algorithm uses directly. We chose to use LTL to represent rules because LTL corresponds closely to natural language and has proven to be a more natural way of expressing tasks and rules for engineers than designing FSAs by hand (Kansou, 2019). Formulae ϕ have the syntax grammar

$$\phi := p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \quad \phi \mid \phi_1 \, \mathcal{U} \, \phi_2$$

where p is a *proposition* (a boolean-valued truth statement that can correspond to objects or events in the world), $\neg$ is negation, $\vee$ is disjunction, is "next", and $\mathcal{U}$ is "until". The derived rules are conjunction ($\wedge$), implication ($\implies$), equivalence ($\leftrightarrow$), "eventually" ($\Diamond\phi \equiv \texttt{True}\,\mathcal{U}\,\phi$) and "always" ($\Box\phi \equiv \neg\Diamond\neg\phi$) (Baier & Katoen, 2008). $\phi_1\,\mathcal{U}\,\phi_2$ means that ϕ_1 is true until ϕ_2 is true, $\Diamond\phi$ means that there is a time where ϕ is true and $\Box\phi$ means that ϕ is always true.

The Options Framework: The options framework is a framework for defining and solving semi-Markov Decision Processes (SMDPs) with a type of macro-action called an option (Sutton et al., 1999). The inclusion of options in an MDP problem turns it into an SMDP problem, because actions are dependent not just on the previous state but also on the identity of the currently active option, which could have been initiated many time steps before the current time.

An option o is a variable-length sequence of actions defined as $o = (\mathcal{I}, \pi, \beta, R_o(s), T_o(s'|s))$. $\mathcal{I} \subseteq \mathcal{S}$ is the initiation set of the option. $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the policy of the option. $\beta : \mathcal{S} \to [0, 1]$ is the termination condition. $R_o(s)$ is the reward model of the option. $T_o(s'|s)$ is the transition model. A major challenge in option learning is that, in general, the number of time steps before the option terminates, k, is a random variable. With this in mind, $R_o(s)$ is defined as the expected cumulative reward of option o given that the option is initiated in state s at time t and ends after k time steps. Letting r_t be the reward received by the agent at t time steps from the beginning of the option,

$$R_o(s) = \mathbb{E}\big[r_1 + \gamma r_2 + \ldots \gamma^{k-1} r_k\big] \tag{1}$$

$T_o(s'|s)$ is the combined probability $p_o(s', k)$ that option o will terminate at state s' after k time steps:

$$T_o(s'|s) = \sum_{k=1}^{\infty} p_o(s', k)\gamma^k \tag{2}$$

A crucial benefit of using options is that they can be composed in arbitrary ways. In the next section, we describe how LOF composes them to satisfy logical specifications.

3. Logical Options Framework

Here is a brief overview of how we will present our formulation of LOF:

1. The LTL formula is decomposed into liveness and safety properties. The liveness property defines the task specification and the safety property defines the costs for violating rules.
2. The propositions are divided into subgoals, safety propositions, and event propositions. Each subgoal is associated with its own option, whose goal is to achieve that subgoal. Safety propositions are used to define rules. Event propositions serve as control flow variables that affect the task.
3. We define an SMDP that is the product of a low-level MDP and a high-level logical FSA.
4. We define the logical options.
5. We present an algorithm for finding the hierarchically optimal policy on the SMDP.
6. We state conditions under which satisfaction of the LTL specification is guaranteed, and we prove that the planning algorithm converges to an optimal policy by showing that the hierarchically optimal SMDP policy is the same as the optimal MDP policy.

The Logic Formula: LTL formulae can be translated into Büchi automata using automatic translation tools such as SPOT (Duret-Lutz et al., 2016). All Büchi automata can be decomposed into liveness and safety properties (Alpern & Schneider, 1987). We assume here that the LTL formula itself can be divided into liveness and safety formulae, $\phi = \phi_{liveness} \wedge \phi_{safety}$. For the case where the LTL formula cannot be factored, see App. A. The liveness property describes "things that must happen" to satisfy the LTL formula. It is a task specification and is used in planning to determine which subgoals the agent must achieve. The safety property describes "things that can never happen" and is used to define costs for violating the rules. In LOF, the liveness property is written using a finite-trace subset of LTL called syntactically co-safe LTL (Bhatia et al., 2010), in which $\Box$ ("always") is not allowed and , $\mathcal{U}$, and $\Diamond$ are only used in positive normal form. This way, the liveness property can be satisfied by finite sequences of propositions, so the property can be represented as an FSA.

Propositions: Propositions are boolean-valued truth statements corresponding to goals, objects, and events in the environment. We distinguish between three types of propositions: subgoals $\mathcal{P}_G$, safety propositions $\mathcal{P}_S$, and event propositions $\mathcal{P}_E$. Subgoals must be achieved in order to satisfy the liveness property. They are associated with goals such as "the agent is at the grocery store". They only appear in $\phi_{liveness}$. Each subgoal may only be associated with one state. Note that in general, it may be impossible to avoid having subgoals appear in ϕ_{safety}. App. A describes how to deal with this scenario. Safety propositions are propositions that the agent must avoid – for example, driving into a lake. They only appear in ϕ_{safety}. Event propositions are not goals, but they can affect the task specification – for example, whether or not a phone call is received. They may occur in $\phi_{liveness}$, and, with extensions described in App. A, in ϕ_{safety}. In the fully observable setting, event propositions are somewhat trivial because the agent knows exactly when/if the event will occur, but in the partially observable setting, they enable complex control flow. Our optimality guarantees only apply in the fully observable setting; however, LOF's properties of satisfaction and composability still apply in the partially observable setting. The goal state of the liveness FSA must be reachable from every other state using only subgoals. This means that no matter what event propositions occur, it must be possible for the agent to satisfy the liveness property. $T_{\mathcal{P}_G} : \mathcal{S} \rightarrow 2^{\mathcal{P}_G}$ and $T_{\mathcal{P}_S} : \mathcal{S} \rightarrow 2^{\mathcal{P}_S}$ relate states to the subgoal and safety propositions that are true at that state. $T_{\mathcal{P}_E} : 2^{\mathcal{P}_E} \rightarrow \{0,1\}$ assigns truth labels to the event propositions.

Hierarchical SMDP: LOF defines a hierarchical semi-Markov Decision Process (SMDP), learns the options, and plans over them. The high level of the SMDP is an FSA specified with LTL. The low level is an environment MDP. We assume that the LTL specification ϕ can be decomposed into a liveness property $\phi_{liveness}$ and a safety property ϕ_{safety}. The propositions $\mathcal{P}$ are the union of the subgoals $\mathcal{P}_G$, safety propositions $\mathcal{P}_S$, and event propositions $\mathcal{P}_E$. We assume that the liveness property can be translated into an FSA $\mathcal{T} = (\mathcal{F}, \mathcal{P}, T_F, R_F, f_0, f_g)$. $\mathcal{F}$ is the set of automaton states; $\mathcal{P}$ is the set of propositions; T_F is the transition function relating the current state and proposition to the next state, $T_F : \mathcal{F} \times \mathcal{P} \times \mathcal{F} \rightarrow [0,1]$. In practice, T_F is deterministic despite our use of probabilistic notation. We assume that there is a single initial state f_0 and final state f_g, and that the goal state f_g is reachable from every state $f \in \mathcal{F}$ using only subgoals. The reward function assigns a reward to every FSA state, $R_F : \mathcal{F} \rightarrow \mathbb{R}$. In our experiments, the safety property takes the form $\bigwedge_{p_s \in \mathcal{P}_S} \Box \neg p_s$, which implies that no safety proposition is allowed, and that they have associated costs, $R_S : 2^{\mathcal{P}_S} \rightarrow \mathbb{R}$. ϕ_{safety} is not limited to this form; App. A covers the general case. There is a low-level environment MDP $\mathcal{E} = (\mathcal{S}, \mathcal{A}, R_{\mathcal{E}}, T_E, \gamma)$. $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space. They can be discrete or continuous. $R_E : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is a low-level reward function that characterizes, for example, distance or actuation costs. $R_{\mathcal{E}}$ is a combination of the safety reward function R_S and R_E, e.g. $R_{\mathcal{E}}(s,a) = R_E(s,a) + R_S(T_{\mathcal{P}_S}(s))$. The transition function of the environment is $T_E : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$.

From these parts we define a hierarchical SMDP $\mathcal{M} = (\mathcal{S} \times \mathcal{F}, \mathcal{A}, \mathcal{P}, \mathcal{O}, T_E \times T_P \times T_F, R_{SMDP}, \gamma)$. The hierarchical state space contains two elements: low-level states $\mathcal{S}$ and FSA states $\mathcal{F}$. The action space is $\mathcal{A}$. The set of propositions is $\mathcal{P}$. The set of options (one option associated with each subgoal in $\mathcal{P}_G$) is $\mathcal{O}$. The transition function consists of the low-level environment transitions T_E and the FSA transitions T_F. $T_P = T_{\mathcal{P}_G} \times T_{\mathcal{P}_S} \times T_{\mathcal{P}_E}$. We call T_P, relating states to propositions, a transition function because it determines when FSA transitions occur. The transitions are applied in the order T_E, T_P, T_F. The reward function

Algorithm 1 Learning and Planning with Logical Options

Given:
Propositions $\mathcal{P}$ partitioned into subgoals $\mathcal{P}_G$, safety propositions $\mathcal{P}_S$, and event propositions $\mathcal{P}_E$
Logical FSA $\mathcal{T} = (\mathcal{F}, \mathcal{P}_G \times \mathcal{P}_E, T_F, R_F, f_0, f_g)$ derived from $\phi_{liveness}$
Low-level MDP $\mathcal{E} = (\mathcal{S}, \mathcal{A}, R_{\mathcal{E}}, T_E, \gamma)$, where $R_{\mathcal{E}}(s,a) = R_E(s,a) + R_S(T_{\mathcal{P}_S}(s))$ combines the environment and safety rewards
Proposition labeling functions $T_{\mathcal{P}_G} : \mathcal{S} \rightarrow 2^{\mathcal{P}_G}$, $T_{\mathcal{P}_S} : \mathcal{S} \rightarrow 2^{\mathcal{P}_S}$, and $T_{\mathcal{P}_E} : 2^{\mathcal{P}_E} \rightarrow \{0,1\}$
To learn:
Set of options $\mathcal{O}$, one for each subgoal $p \in \mathcal{P}_G$
Meta-policy $\mu(f,s,o)$, $Q(f,s,o)$, and $V(f,s)$
Learn logical options:
for $p \in \mathcal{P}_G$ **do**
    Learn an option that achieves p,
    $o_p = (\mathcal{I}_{o_p}, \pi_{o_p}, \beta_{o_p}, R_{o_p}(s), T_{o_p}(s'|s))$
    $\mathcal{I}_{o_p} = \mathcal{S}$
    $\beta_{o_p} = \begin{cases} 1 & \text{if } p \in T_{\mathcal{P}_G}(s) \\ 0 & \text{otherwise} \end{cases}$
    $\pi_{o_p} =$ optimal policy on $\mathcal{E}$ with rollouts terminating when $p \in T_{\mathcal{P}_G}(s)$
    $T_{o_p}(s'|s) = \begin{cases} \mathbb{E}\gamma^k & \text{if } p \in T_{\mathcal{P}_G}(s')\text{; } k \text{ is number of time steps to reach } p \\ 0 & \text{otherwise} \end{cases}$
    $R_{o_p}(s) = \mathbb{E}[R_{\mathcal{E}}(s,a_1) + \gamma R_{\mathcal{E}}(s_1,a_2) + \dots + \gamma^{k-1} R_{\mathcal{E}}(s_{k-1},a_k)]$
end for
Find a meta-policy μ over the options:
Initialize $Q : \mathcal{F} \times \mathcal{S} \times \mathcal{O} \rightarrow \mathbb{R}$, $V : \mathcal{F} \times \mathcal{S} \rightarrow \mathbb{R}$ to 0
for $(k,f,s) \in [1,\dots,n] \times \mathcal{F} \times \mathcal{S}$ **do**
    for $o \in \mathcal{O}$ **do**
        $Q_k(f,s,o) \leftarrow R_F(f)R_o(s) + \sum_{f' \in \mathcal{F}} \sum_{\bar{p}_e \in 2^{\mathcal{P}_E}} \sum_{s' \in \mathcal{S}} T_F(f'|f, T_P(s'), \bar{p}_e) T_{P_E}(\bar{p}_e) T_o(s'|s) V_{k-1}(f',s')$
    end for
    $V_k(f,s) \leftarrow \max_{o \in \mathcal{O}} Q_k(f,s,o)$
end for
$\mu(f,s,o) = \arg\max_{o \in \mathcal{O}} Q(f,s,o)$
Return: Options $\mathcal{O}$, meta-policy $\mu(f,s,o)$ and Q- and value functions $Q(f,s,o), V(f,s)$

$R_{SMDP}(f,s,o) = R_F(f)R_o(s)$, so $R_F(f)$ is a weighting on the option rewards. The SMDP has the same discount factor γ as $\mathcal{E}$. Planning is done on the SMDP in two steps: first, the options $\mathcal{O}$ are learned over $\mathcal{E}$ using an appropriate policy-learning algorithm such as PPO or Reward Machines. Next, a meta-policy over the task specification $\mathcal{T}$ is found using the learned options and the reward function R_{SMDP}.

Logical Options: The first step of Alg. 1 is to learn the logical options. We associate every subgoal p with an option $o_p = (\mathcal{I}_{o_p}, \pi_{o_p}, \beta_{o_p}, R_{o_p}, T_{o_p})$. These terms are defined starting at Alg. 1 line 5. One assumption we make is that the initiation set of every option is the entire state space $\mathcal{S}$. This assumption can be easily loosened as long as the liveness property does not require an infeasible sequence of options, e.g., if the task is to go to Room A and then Room B, the initiation set of the "Room B" option must include Room A. Every o_p has a policy π_{o_p} whose goal is to reach the state s_p where p is true. Options are learned by training on the environment MDP $\mathcal{E}$ and terminating only when s_p is reached. As we discuss in Sec. 3.1, under certain conditions the optimal option policy is guaranteed to always terminate at the subgoal. This allows us to simplify the transition model of Eq. 2 to the form in Alg. 1 line 11. In the experiments, we further simplify this expression by setting $\gamma = 1$.

Logical Value Iteration: After finding the logical options, the next step is to find a meta-policy for FSA $\mathcal{T}$ over the options (see Alg. 1 line 14). Q- and value functions are found for the SMDP using the Bellman update equations:

$$Q_k(f, s, o) \leftarrow R_F(f) R_o(s) + \sum_{f' \in \mathcal{F}} \sum_{\bar{p}_e \in 2^{\mathcal{P}_E}} \sum_{s' \in \mathcal{S}} T_F(f'|f, T_{P_G}(s'), \bar{p}_e) T_{P_E}(\bar{p}_e) T_o(s'|s) V_{k-1}(f', s') \tag{3}$$

$$V_k(f, s) \leftarrow \max_{o \in \mathcal{O}} Q_k(f, s, o) \tag{4}$$

Eq. 3 differs from the generic equations for SMDP value iteration in that the transition function has two extra components, $\sum_{f' \in \mathcal{F}} T_F(f'|f, T_P(s'), \bar{p}_e)$ and $\sum_{\bar{p}_e \in 2^{\mathcal{P}_E}} T_{P_E}(\bar{p}_e)$. The equations are derived from Araki et al. (2019) and the fact that, on every step in the environment, three transitions are applied: the option transition T_o, the event proposition "transition" T_{P_E}, and the FSA transition T_F. Note that $R_o(s)$ and $T_o(s'|s)$ compress the consequences of choosing an option o at a state s from a multi-step trajectory into two real-valued numbers, allowing for more efficient planning.

3.1. Conditions for Satisfaction and Optimality

Here we give an overview of the proofs and necessary conditions for satisfaction and optimality. The full proofs and definitions are in App. B.

First, we describe the condition for an optimal option to always reach its subgoal. Let $\pi'(s|s')$ be the optimal goal-conditioned policy for reaching a goal s'. If the optimal option policy equals the goal-conditioned policy for reaching the subgoal s_g, i.e. $\pi^*(s) = \pi_g(s|s_g)$, then the option will always reach the subgoal. This can be stated in terms of value functions: let $V^{\pi'}(s|s')$ be the expected return of $\pi'(s|s')$. If $V^{\pi_g}(s|s_g) > V^{\pi'}(s|s')\ \forall s, s' \neq s_g$, then $\pi^*(s) = \pi_g(s|s_g)$. This occurs for example if $-\infty < R_{\mathcal{E}}(s, a) < 0$ and if the episode terminates when the agent reaches s_g. Then V^{π_g} is a bounded negative number, and $V^{\pi'}$ for all other states is $-\infty$. We show that if every option is guaranteed to achieve its subgoal, then there must exist at least one sequence of options that satisfies the specification.

We then give the condition for the hierarchically optimal meta-policy $\mu^*(s)$ to always achieve the FSA goal state f_g. In our context, hierarchical optimality means that the meta-policy is optimal over the available options. Let $\mu'(f, s|f')$ be the hierarchically optimal goal-conditioned meta-policy for reaching FSA state f'. If the hierarchically optimal meta-policy equals the goal-conditioned meta-policy for reaching the FSA goal state f_g, i.e. $\mu^*(f, s) = \mu_g(f, s|f_g)$, then $\mu^*(f, s)$ will always reach f_g. In terms of value functions: let $V^{\mu'}(f, s|f')$ be the expected return for μ'. If $V^{\mu_g}(f, s|f_g) > V^{\mu'}(f, s|f') \forall f, s, f' \neq f_g$, then $\mu^* = \mu_g$. This occurs if all FSA rewards $R_F(f) > 0$, all environment rewards $-\infty < R_{\mathcal{E}}(s, a) < 0$, and the episode only terminates when the agent reaches f_g. Then V^{μ_g} is a bounded negative number, and $V^{\mu'}$ for all other states is $-\infty$. Because LOF uses the Bellman update equations to learn the meta-policy, the LOF meta-policy will converge to the hierarchically optimal meta-policy.

Consider the SMDP where planning is allowed over low-level actions, and let us call it the "hierarchical MDP" (HMDP) with optimal policy π^*_{HMDP}. Our result is:

Theorem 3.1. *Given that the conditions for satisfaction and hierarchical optimality are met, the LOF hierarchically optimal meta-policy μ_g with optimal option sub-policies π_g has the same expected returns as the optimal policy π^*_{HMDP} and satisfies the task specification.*

3.2. Composability

The results in Sec. 3.1 guarantee that LOF's learned model can be composed to satisfy new tasks. Furthermore, the composed policy has the same properties as the original policy – satisfaction and optimality. LOF's possession of composability along with satisfaction and optimality derives from two facts: 1) Options are inherently composable because they can be executed in any order. 2) If the conditions of Thm. 3.1 are met, LOF is guaranteed to find a (hierarchically) optimal policy over the options that will satisfy any liveness property that uses subgoals associated with the options. The composability of LOF distinguishes it from other algorithms that can achieve satisfaction and optimality.

4. Experiments & Results

Experiments: We performed experiments to demonstrate satisfaction and composability [1]. For satisfaction, we measure cumulative reward over training steps. Cumulative reward is a proxy for satisfaction, as the environments can only achieve the maximum reward when they satisfy their tasks. For the composability experiments, we take the trained options and record how many meta-policy retraining steps it takes to learn an optimal meta-policy for a new task.

Environments: We measure the performance of LOF on three environments. The first environment is a discrete gridworld (Fig. 3a) called the "delivery domain," as it can represent a delivery truck delivering packages to three locations (a, b, c) and having a home base h. There are also obstacles o (the black squares). The second environment is called the reacher domain, from OpenAI Gym (Fig. 3d). It is a two-link arm that has continuous state and action spaces. There are four subgoals represented by colored balls: red r, green g, blue b, and yellow y. The third environment is called the pick-and-place domain, and it is a continuous 3D environment with a robotic Panda arm from CoppeliaSim and PyRep (James et al., 2019). It is inspired by the lunchbox-packing experiments of Araki et al. (2019) in which subgoals r, g, and b are food items that must be packed into lunchbox y. All environments also have an event proposition called can, which represents when the need to fulfill part of a task is cancelled.

Tasks: We test satisfaction and composability on four tasks. The first task is a "sequential" task. For the delivery domain, the LTL formula is $\diamondsuit(a \wedge \diamondsuit(b \wedge \diamondsuit(c \wedge \diamondsuit h))) \wedge \Box\neg o$ – "deliver package a, then b, then c, and then return to home h. And always avoid obstacles." The next task is the "IF" task (equivalent to the task shown in Fig. 1b): $(\diamondsuit(c \wedge \diamondsuit a) \wedge \Box\neg can) \vee (\diamondsuit c \wedge \diamondsuit can) \wedge \Box\neg o$ – "deliver package c, and then a, unless a gets cancelled. And always avoid obstacles". We call the third task the "OR" task, $\diamondsuit((a \vee b) \wedge \diamondsuit c) \wedge \Box\neg o$ – "deliver package a or b, then c, and always avoid obstacles". The "composite" task has elements of all three of the previous tasks: $(\diamondsuit((a \vee b) \wedge \diamondsuit(c \wedge \diamondsuit h)) \wedge \Box\neg can) \vee (\diamondsuit((a \vee b) \wedge \diamondsuit h) \wedge \diamondsuit can) \wedge \Box\neg o$. "Deliver package a or b, and then c, unless c gets cancelled, and then return to home h. And always avoid obstacles". The tasks for the reacher and pick-and-place environments are equivalent, except that there are no obstacles for the reacher and arm to avoid.

The sequential task is meant to show that planning is efficient and effective even for long-time horizon tasks. The "IF" task shows that the agent's policy can respond to event propositions, such as being alerted that a delivery is cancelled. The "OR" task is meant to demonstrate the optimality of our algorithm versus a greedy algorithm, as discussed in Fig. 2. Lastly, the composite task shows that learning and planning are efficient and effective even for complex tasks.

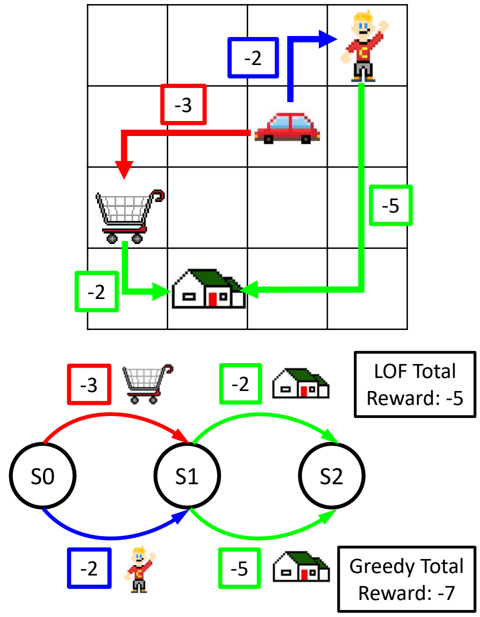
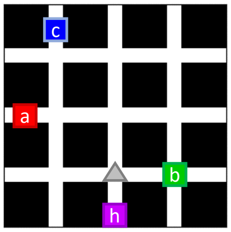


Figure 2. In this environment, the agent must either pick up the kid or go grocery shopping, and then go home (the "OR" task). Starting at $S0$, the greedy algorithm picks the next step in the FSA with the lowest cost (picking up the kid), which leads to a higher overall cost. LOF finds the optimal path through the FSA.

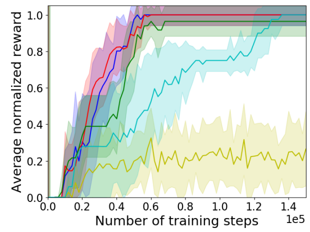
Baselines: We test four baselines against our algorithm. Our algorithm is `LOF-VI`, short for "Logical Options Framework with Value Iteration," because it uses value iteration for high-level planning. `LOF-QL` uses Q-learning instead (details are in App. C.3). Unlike `LOF-VI`, `LOF-QL` does not need explicit knowledge of T_F, the FSA transition function. `Greedy` is a naive implementation of task satisfaction; it uses its knowledge of the FSA to select the next subgoal with the lowest cost to attain. This leaves it vulnerable to choosing suboptimal paths through the FSA, as shown in Fig. 2. `Flat Options` uses the options framework with no knowledge of the FSA. Its SMDP formulation is not hierarchical – the state space and transition function do not contain high-level states $\mathcal{F}$ or transition function T_F. The last baseline is `RM`, short for Q-Learning for Reward Machines (Icarte et al., 2018). Whereas LOF learn options to accomplish subgoals, `RM` learns sub-policies for every FSA state. App. C.4 discusses the differences between `RM` and LOF in detail.

Implementation: For the delivery domain, options were learned using Q-learning with an ϵ-greedy exploration policy. `RM` was learned using the Q-Learning for Reward Ma-

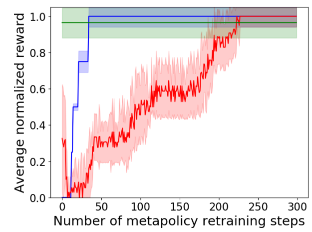
[1]Code for the discrete domain experiments is available at https://github.com/braraki/logical-options-framework. Code for the other domains is available in the supplementary material.



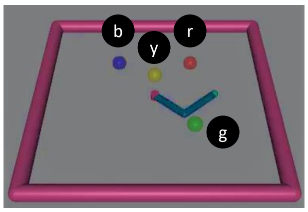
(a) Delivery domain.



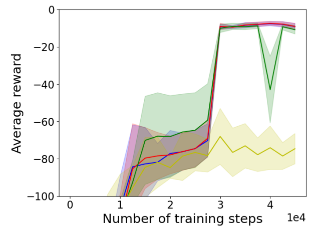
(b) Satisfaction performance.



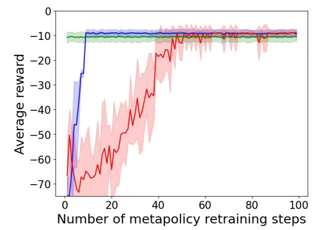
(c) Composability performance.



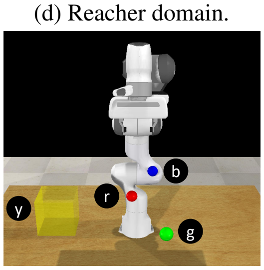
(d) Reacher domain.



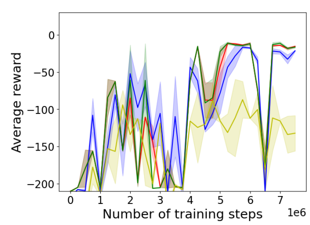
(e) Satisfaction performance.



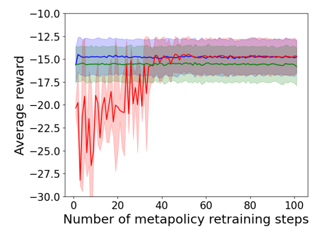
(f) Composability performance.



(g) Pick-and-place domain.



(h) Satisfaction performance.



(i) Composability performance.

LOF-VI LOF-QL Greedy Flat Options Reward Machines

Figure 3. Performance on the satisfaction and composability experiments, averaged over all tasks. Note that `LOF-VI` composes new meta-policies in just 10-50 retraining steps. The first row is the delivery domain, the second row is the reacher domain, and the third row is the pick-and-place domain. All results, including `RM` performance on the reacher and pick-and-place domains, are in App. C.6.

chines (QRM) algorithm described in Icarte et al. (2018). For the reacher and pick-and-place domains, options were learned by using proximal policy optimization (PPO) (Schulman et al., 2017) to train goal-oriented policy and value functions, which were represented using 128×128 and $128 \times 128 \times 128$ fully connected neural networks, respectively. Deep-QRM was used to train `RM`. The implementation details are discussed more fully in App. C.

4.1. Results

Satisfaction: Results for the satisfaction experiments, averaged over all four tasks, are shown in Figs. 3b, 3e, and 3h. (Results on all tasks are in App. C.6). As expected, `Flat Options` shows no ability to satisfy tasks, as it has no knowledge of the FSAs. `Greedy` trains as quickly as `LOF-VI` and `LOF-QL`, but its returns plateau before the others because it chooses suboptimal paths in the composite and OR tasks. The difference is small in the continuous domains but still present. `LOF-QL` achieves as high a return as `LOF-VI`, but it is less composable (discussed below). `RM` learns much more slowly than the other methods. This is because for `RM`, a reward is only given for reaching the goal state, whereas in the LOF-based methods, options are rewarded for reaching their subgoals, so during training LOF-based methods have a richer reward function than `RM`. For the continuous domains, `RM` takes an order of magnitude more steps to train, so we left it out of the figures for clarity (see App. Figs. 14 and 16). However, in the continuous domains, `RM` eventually achieves a higher return than the LOF-based methods. This is because for those domains, we define the subgoals to be spherical regions rather than single states, violating one of the conditions for optimality. Therefore, for example, it is possible that the meta-policy does not take advantage of the dynamics of the arm to swing through the subgoals more efficiently. `RM` does not have this condition and learns a single policy that can take advantage of inter-subgoal dynamics to learn a more optimal policy.

Composability: The composability experiments were done on the three composable baselines, `LOF-VI`, `LOF-QL`, and `Greedy`. App. C.4 discusses why `RM` is not composable. `Flat Options` is not composable because its formulation does not include the FSA $\mathcal{T}$. Therefore it is completely incapable of recognizing and adjusting to changes in the FSA. The composability results are shown in Figs. 3c, 3f, and 3i. `Greedy` requires no retraining steps to "learn" a meta-policy on a new FSA – given its current FSA state, it simply chooses the next available FSA state that has the lowest cost to achieve. However, its meta-policy may be arbitrarily suboptimal. `LOF-QL` learns optimal (or in the continuous case, close-to-optimal) policies, but it takes $\sim$50-250 retraining steps, versus $\sim$10-50 for `LOF-VI`. Therefore `LOF-VI` strikes a balance between `Greedy` and `LOF-QL`, requiring far fewer steps than `LOF-QL` to retrain, and achieving better performance than `Greedy`.

5. Related Work

We distinguish our work from related work in HRL by its possession of three desirable properties – composability, satisfaction, and optimality. Most other works possess two of these properties at the cost of the other.

Not Composable: The previous work most similar to ours is Icarte et al. (2018), which introduces a framework to solve tasks defined by automata called Reward Machines. Their algorithm, Q-Learning for Reward Machines, learns a sub-policy for every state of the automaton that achieves satisfaction and optimality. However, the learned sub-policies have limited composability because they end up learning a specific path through the automaton, and if the structure of the automaton is changed, there is no guarantee that the sub-policies will be able to satisfy the new automaton without re-training. By contrast, LOF learns a sub-policy for every subgoal, independent of the automaton, and therefore the sub-policies can be arranged to satisfy arbitrary tasks. Another similar work is Logical Value Iteration (LVI) (Araki et al., 2019; 2020). LVI defines a hierarchical MDP and value iteration equations that find satisfying and optimal policies; however, the algorithm is limited to discrete domains and has limited composability. A number of HRL algorithms use reward shaping to guide the agent through the states of an automaton (Li et al., 2017; 2019; Camacho et al., 2019; Hasanbeig et al., 2018; Jothimurugan et al., 2019; Shah et al., 2020; Yuan et al., 2019; De Giacomo et al., 2019). While these algorithms can guarantee satisfaction and sometimes optimality, they cannot be composed because their policies are not hierarchical. Another approach is to use a symbolic planner to find a satisfying sequence of tasks and use an RL agent to learn and execute that sequence of tasks (Gordon et al., 2019; Illanes et al., 2020; Lyu et al., 2019). However, the meta-controllers of Gordon et al. (2019) and Lyu et al. (2019) are not composable as they are trained together with the low-level controllers. Although the work of Illanes et al. (2020) is amenable to transfer learning, it is not composable. Paxton et al. (2017); Mason et al. (2017) use logical constraints to guide exploration, and while these approaches are also satisfying and optimal, they are not composable as the agent is trained for a specific set of rules. LOF is composable unlike the above methods because it has a hierarchical action space with high-level options. Once the options are learned, they can be composed arbitrarily.

Not Satisfying: Most hierarchical frameworks cannot satisfy tasks. Instead, they focus on using state and action abstractions to make learning more efficient (Dietterich, 2000; Dayan & Hinton, 1993; Parr & Russell, 1998; Diuk et al., 2008; Oh et al., 2019). The options framework (Sutton et al., 1999) stands out because of its composability and its guarantee of hierarchical optimality, which is why we based our work off of it. There is also a class of HRL algorithms that builds on the idea of goal-oriented policies that can navigate to nearby subgoals (Eysenbach et al., 2019; Ghosh et al., 2018; Faust et al., 2018). By sampling sequences of subgoals and using a goal-oriented policy to navigate between them, these algorithms can travel much longer distances than a policy can travel on its own. Although these algorithms are "composable" in that they can navigate to far-away goals without further training, they are not able to solve tasks. Andreas et al. (2017) present an algorithm for solving simple policy "sketches" which is also composable; however, sketches are considerably less expressive than automata and linear temporal logic, which we use. Unlike the above methods, LOF is satisfying because it has a hierarchical state space with low-level MDP states and high-level FSA states. Therefore LOF can satisfy tasks by learning policies that reach the FSA goal state.

Not Optimal: In HRL, there are at least three types of optimality – hierarchical, recursive, and overall. As defined in Dietterich (2000), the hierarchically optimal policy is the optimal policy given the constraints of the hierarchy, and recursive optimality is when a policy is optimal given the policies of its children. For example, the options framework is hierarchically optimal, while MAXQ and abstract MDPs (Gopalan et al., 2017) are recursively optimal. Icarte et al. (2019) introduce a composable method for learning Reward Machines, but their approach is focused on the reinforcement learning setting and is only guaranteed to learn hierarchically optimal policies. Leon et al. (2020) introduce a neurosymbolic method for generating policies for satisfying logical specifications that can zero-shot generalize to new specifications. However, the "symbolic module" of their network has no guarantees on being able to optimally satisfy specifications. The method described in Kuo et al. (2020) is fully composable, but not optimal as it uses a recurrent neural network to generate a sequence of high-level actions and is therefore not guaranteed to find optimal policies. The approach in Kuo et al. (2020) has some advantages in terms of scalability versus our approach, since they use a single deep model to learn policies for all operators and propositions. However, for LOF, a single deep goal-oriented policy could be trained to reach all subgoals, as is done in Leon et al. (2020). We therefore believe that LOF could be made equally scalable as Kuo et al. (2020), although this would be at the cost of losing most of the guarantees of the framework. However, one guarantee that LOF would keep is the hierarchical optimality of the meta-policy, which is a result of finding the meta-policy using value iteration. We therefore believe that LOF should have better performance on unseen specifications than Kuo et al. (2020), because Kuo et al. (2020) has no guarantees for satisfying specifications. LOF is hierarchically optimal because it finds an optimal meta-policy over the high-level options, and as we state in the paper, there are also conditions under which the overall policy is optimal.

6. Discussion and Conclusion

In this work, we claim that LOF has a unique combination of three properties: satisfaction, optimality, and composability. We state and prove the conditions for satisfaction and optimality in Sec. 3.1. The experimental results confirm our claims while also pointing out some weaknesses. `LOF-VI` achieves optimal or near-optimal policies and trains an order of magnitude faster than the existing work most similar to it, `RM`. However, the optimality condition that each subgoal be associated with one state cannot be met for continuous domains, and therefore `RM` eventually outperforms `LOF-VI`. But even when optimality is not guaranteed, `LOF-VI` is hierarchically optimal, which is why it outperforms `Greedy` in the composite and OR tasks. Next, the composability experiments show that `LOF-VI` can compose its learned options to perform new tasks in about 10-50 iterations on the benchmarks. Although `Greedy` requires no retraining steps, 10-50 retraining iterations is a tiny fraction of the tens of thousands of steps required to learn the original policy. Lastly, we have shown that LOF learns policies efficiently, and that it can be used with a variety of domains and policy-learning algorithms.

Acknowledgments

This material is supported by the Toyota Research Institute within the Toyota-CSAIL joint research center, NSF grant IIS-1723943, ONR N00014-18-1-2830, the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001, and SUTD Project Astralis. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

References

Abel, D. and Winder, J. The expected-length model of options. In *IJCAI*, 2019.

Alpern, B. and Schneider, F. B. Recognizing safety and liveness. *Distributed computing*, 2(3):117–126, 1987.

Andreas, J., Klein, D., and Levine, S. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, pp. 166–175, 2017.

Araki, B., Vodrahalli, K., Leech, T., Vasile, C. I., Donahue, M., and Rus, D. Learning to plan with logical automata. In *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, June 2019. doi: 10.15607/RSS.2019.XV.064.

Araki, B., Vodrahalli, K., Leech, T., Vasile, C. I., Donahue, M., and Rus, D. Deep bayesian nonparametric learning of rules and plans from demonstrations with a learned automaton prior. In *AAAI*, pp. 10026–10034, 2020.

Baier, C. and Katoen, J. *Principles of model checking*. MIT Press, 2008. ISBN 978-0-262-02649-9.

Bhatia, A., Kavraki, L. E., and Vardi, M. Y. Sampling-based motion planning with temporal goals. In *2010 IEEE International Conference on Robotics and Automation*, pp. 2689–2696. IEEE, 2010.

Camacho, A., Icarte, R. T., Klassen, T. Q., Valenzano, R. A., and McIlraith, S. A. Ltl and beyond: Formal languages for reward function specification in reinforcement learning. In *IJCAI*, volume 19, pp. 6065–6073, 2019.

Clarke, E. M., Grumberg, O., and Peled, D. *Model Checking*. MIT Press, 2001. ISBN 978-0-262-03270-4.

Dayan, P. and Hinton, G. E. Feudal reinforcement learning. In *Advances in neural information processing systems*, pp. 271–278, 1993.

De Giacomo, G., Iocchi, L., Favorito, M., and Patrizi, F. Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pp. 128–136, 2019.

Dietterich, T. G. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000.

Diuk, C., Cohen, A., and Littman, M. L. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pp. 240–247, 2008.

Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., and Xu, L. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16)*, volume 9938 of *Lecture Notes in Computer Science*, pp. 122–129. Springer, October 2016. doi: 10.1007/978-3-319-46520-3_8.

Eysenbach, B., Salakhutdinov, R. R., and Levine, S. Search on the replay buffer: Bridging planning and reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 15220–15231, 2019.

Faust, A., Oslund, K., Ramirez, O., Francis, A., Tapia, L., Fiser, M., and Davidson, J. Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5113–5120. IEEE, 2018.

Ghosh, D., Gupta, A., and Levine, S. Learning actionable representations with goal-conditioned policies. *arXiv preprint arXiv:1811.07819*, 2018.

Gopalan, N., Littman, M. L., MacGlashan, J., Squire, S., Tellex, S., Winder, J., Wong, L. L., et al. Planning with abstract markov decision processes. In *Twenty-Seventh International Conference on Automated Planning and Scheduling*, 2017.

Gordon, D., Fox, D., and Farhadi, A. What should i do now? marrying reinforcement learning and symbolic planning. *arXiv preprint arXiv:1901.01492*, 2019.

Hasanbeig, M., Abate, A., and Kroening, D. Logically-constrained reinforcement learning. *arXiv preprint arXiv:1801.08099*, 2018.

Icarte, R. T., Klassen, T., Valenzano, R., and McIlraith, S. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, pp. 2107–2116, 2018.

Icarte, R. T., Waldie, E., Klassen, T., Valenzano, R., Castro, M., and McIlraith, S. Learning reward machines for partially observable reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 15523–15534, 2019.

Illanes, L., Yan, X., Icarte, R. T., and McIlraith, S. A. Symbolic plans as high-level instructions for reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pp. 540–550, 2020.

James, S., Freese, M., and Davison, A. J. Pyrep: Bringing v-rep to deep robot learning. *arXiv preprint arXiv:1906.11176*, 2019.

Jothimurugan, K., Alur, R., and Bastani, O. A composable specification language for reinforcement learning tasks. In *Advances in Neural Information Processing Systems*, pp. 13041–13051, 2019.

Kansou, B. K. A. Converting asubset of ltl formula to buchi automata. *International Journal of Software Engineering & Applications (IJSEA)*, 10(2), 2019.

Kuo, Y.-L., Katz, B., and Barbu, A. Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of ltl formulas. *arXiv preprint arXiv:2006.01110*, 2020.

Leon, B. G., Shanahan, M., and Belardinelli, F. Systematic generalisation through task temporal logic and deep reinforcement learning. *arXiv preprint arXiv:2006.08767*, 2020.

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

Li, X., Serlin, Z., Yang, G., and Belta, C. A formal methods approach to interpretable reinforcement learning for robotic planning. *Science Robotics*, 4(37), 2019.

Lyu, D., Yang, F., Liu, B., and Gustafson, S. Sdrl: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 2970–2977, 2019.

Mason, G. R., Calinescu, R. C., Kudenko, D., and Banks, A. Assured reinforcement learning with formally verified abstract policies. In *9th International Conference on Agents and Artificial Intelligence (ICAART)*. York, 2017.

Oh, Y., Patel, R., Nguyen, T., Huang, B., Pavlick, E., and Tellex, S. Planning with state abstractions for non-markovian task specifications. *arXiv preprint arXiv:1905.12096*, 2019.

Parr, R. and Russell, S. J. Reinforcement learning with hierarchies of machines. In *Advances in neural information processing systems*, pp. 1043–1049, 1998.

Paxton, C., Raman, V., Hager, G. D., and Kobilarov, M. Combining neural networks and tree search for task and motion planning in challenging environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6059–6066. IEEE, 2017.

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

Shah, A., Li, S., and Shah, J. Planning with uncertain specifications (puns). *IEEE Robotics and Automation Letters*, 5(2):3414–3421, 2020.

Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

Yuan, L. Z., Hasanbeig, M., Abate, A., and Kroening, D. Modular deep reinforcement learning with temporal logic specifications. *arXiv preprint arXiv:1909.11591*, 2019.

Zhang, S. and Sridharan, M. A survey of knowledge-based sequential decision making under uncertainty. *arXiv preprint arXiv:2008.08548*, 2020.