LOGICAL OPTIONS FRAMEWORK

Anonymous authors
Paper under double-blind review

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 no additional learning. We evaluate LOF on four tasks in discrete and continuous domains.

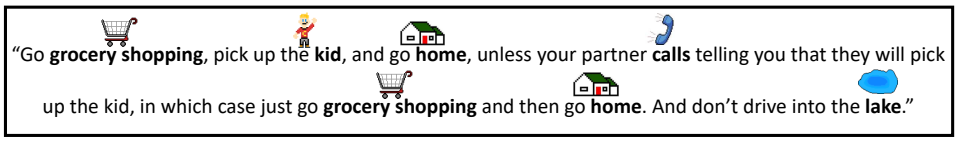
1 INTRODUCTION

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 (Marcus, 2020). Many environments obey rules that can be represented as logical formulae; imagine, for example, the rules a car follows while driving, or a recipe a chef follows to cook a dish. Traditional motion and path planning techniques struggle to formulate plans over these kinds of 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). Most of today's algorithms sacrifice at least one of these objectives. For example, Reward Machines from Icarte et al. (2018) is satisfying and optimal, but not composable; the options framework (Sutton et al., 1999) is composable and hierarchically optimal, but cannot satisfy specifications. We introduce a new approach called 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 show that our framework is compatible with a large variety of domains and planning algorithms, from discrete domains and value iteration to continuous domains and proximal policy optimization (PPO).

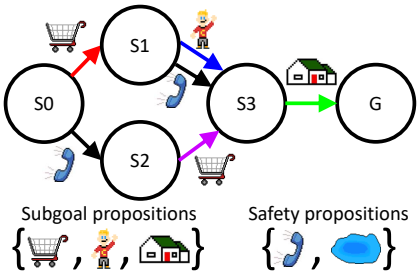
Satisfaction: An agent operating in an environment governed by rules must be able to satisfy the specified rules. Satisfaction is a concept from formal logic, in which the input to a logical formula causes the formula to evaluate to `True`. Logical formulae can encapsulate rules and tasks like the ones 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.

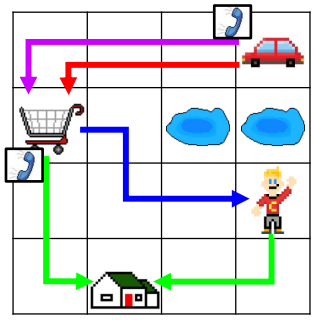
Composability: Our method also has the property of composability – once it has learned to satisfy a task, the learned model can be rearranged to satisfy a large variety of related tasks. A shortcoming of many RL models is that they are not composable – trained to solve one specific task, they are incapable of handling even small variations in the task structure. However, the real world is a dynamic and unpredictable place, so the ability to automatically reason over as-yet-unseen tasks and rules is a crucial element of intelligence.



(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" and "safety" propositions.



(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. "Subgoal" propositions correspond to tasks that the agent can fulfill. "Safety" propositions correspond either to events that the agent has no control over or to events that they must avoid. The arrows in (c) represent subpolicies, 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 subpolicies can occur only after the agent receives the phone call.

The illustrations in Fig. 1 give an overview of our work. 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, and "safety" propositions, representing events that you have no control over (getting a phone call) or must avoid (driving into a lake) (Fig. 1b). In this environment, you have to follow rules (Fig. 1a). These rules can be converted into a logic formula, and from there into a finite state automaton (FSA) (Fig. 1b). The Logical Options Framework learns an option for each subgoal (illustrated by the arrows in Fig. 1c), and a metapolicy for choosing amongst the options to reach the goal state of the FSA. After learning, the options can be recombined to fulfill other tasks.

1.1 Contributions

We introduce the Logical Options Framework (LOF), which 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 metapolicies for the SMDP that allows for the options to be composed to solve new tasks with only 10-50 retraining steps.
3. Conditions and proofs for achieving satisfaction and optimality.
4. Experiments on a discrete domain and a continuous domain on four tasks demonstrating satisfaction, optimality, and composability.

2 Background

Linear Temporal Logic: We use linear temporal logic (LTL) to formally specify tasks (Clarke et al., 2001). Formulae ϕ have the syntax grammar

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

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, $\bigcirc$ 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 or subpolicy 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. $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ is the policy the option follows while the option is active, $\beta : \mathcal{S} \rightarrow [0, 1]$ is the termination condition, $R_o(s)$ is the reward model of the option, and $T_o(s'|s)$ is the transition model. $R_o(s)$ 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.

$$R_o(s) = \mathbb{E}\big[r_{t+1} + \gamma r_{t+2} + \ldots \gamma^{k-1} r_{t+k}\big] \tag{1}$$

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

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

3 Logical Options Framework

The Logic Formula: LTL formulae can be represented as Büchi automata, which can be decomposed into liveness and safety properties (Alpern & Schneider, 1987): $\phi = \phi_{liveness} \wedge \phi_{safety}$. The liveness property describes "things that must happen" to satisfy the LTL formula. It is a task specification, and it 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 must be written using a finite-trace subset of LTL called syntactically co-safe LTL (Bhatia et al., 2010), in which the $\Box$ ("always") operator is not allowed and $\bigcirc$, $\mathcal{U}$, and $\Diamond$ are only used in positive normal form. This way, the liveness property can be satisfied by finite-length sequences of propositions, and 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 assume that every state is associated with a single proposition as defined by a labeling function $T_P : \mathcal{S} \rightarrow \mathcal{P}$. We choose to limit each state to have only one proposition because the number of possible combinations of propositions would otherwise be $2^{|\mathcal{P}|}$. Instead, if a combination of propositions does occur, a new conjugate proposition can be defined. We assume that there are two types of propositions – subgoals $\mathcal{P}_G$ and safety propositions $\mathcal{P}_S$. Subgoal propositions are propositions associated with goals such as "the agent is at the grocery store" that the agent must seek out. These may only appear in the liveness property $\phi_{liveness}$, and there may only be one instance of each subgoal. Safety propositions are propositions such as "the agent received a call" that the agent has no control over or does not seek out. These may appear in both the liveness and safety properties. Safety propositions are used to define the costs for violating safety rules.

Logical Options: We associate every subgoal p with an option $o_p = (\mathcal{I}_{o_p}, \pi_{o_p}, \beta_{o_p}, R_{o_p}, T_{o_p})$. Every o_p has a policy π_{o_p} whose goal is to reach the state s_p where p is true. Options can be learned by training on the environment MDP $\mathcal{E}$ and terminating training 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. The components of the option are defined in Alg. 1 lines 6-12.

Hierarchical SMDP: We assume that 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 either discrete or continuous. $R_\mathcal{E}$ is the low-level reward function, $R_\mathcal{E} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. T_E is the transition function of the environment, $T_E : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$.

Algorithm 1 Learning and Planning with Logical Options

procedure LEARNING-AND-PLANNING-WITH-LOGICAL-OPTIONS
  Given:
    LTL specification $\phi = \phi_{liveness} \wedge \phi_{safety}$ with propositions $\mathcal{P}$ partitioned into subgoals $\mathcal{P}_G$ and safety propositions $\mathcal{P}_S$
    Low-level MDP $\mathcal{E} = (\mathcal{S}, \mathcal{A}, R_{\mathcal{E}}, T_E, \gamma)$
    Logical automaton $\mathcal{T} = (\mathcal{F}, \mathcal{P}, T_F, R_F, f_0, f_g)$ derived from $\phi_{liveness}$
    Proposition labeling function $T_P : \mathcal{S} \rightarrow \mathcal{P}$
  To learn:
    Set of options $\mathcal{O}$, one for each subgoal proposition $p \in \mathcal{P}_G$
    Metapolicy $\mu(f, s, o)$ along with $Q(f, s, o)$ and $V(f, s)$
  Learn logical options:
  For every p in $\mathcal{P}_G$, learn an option for achieving p, $o_p = (\mathcal{I}_{o_p}, \pi_{o_p}, \beta_{o_p}, R_{o_p}(s), T_{o_p}(s'|s))$
    $\mathcal{I}_{o_p} = \mathcal{S}$
    $\beta_{o_p} = \begin{cases} 1 & \text{if } T_P(s, p) = 1 \\ 0 & \text{otherwise} \end{cases}$
    π_{o_p} = optimal policy on $\mathcal{E}$ with rollouts terminating when $T_P(s) = p$
    $T_{o_p}(s'|s) = \begin{cases} \gamma^k & \text{if } T_P(s') = p, \text{where } k \text{ is number of time steps to reach } p \\ 0 & \text{otherwise} \end{cases}$
    $R_{o_p}(s) = \mathbb{E}[\mathcal{R}_{\mathcal{E}}(s, a) + \gamma \mathcal{R}_{\mathcal{E}}(s', a') + \gamma^2 \mathcal{R}_{\mathcal{E}}(s'', a'') + \dots]$
  Find a metapolicy μ over the options:
  Initialize $Q : \mathcal{F} \times \mathcal{S} \times \mathcal{O} \rightarrow \mathbb{R}$ and $V : \mathcal{F} \times \mathcal{S} \rightarrow \mathbb{R}$ to 0
  For $(k, f, s) \in [1, \dots, n] \times \mathcal{F} \times \mathcal{S}$:
    For $o \in \mathcal{O}$:
      $Q_k(f, s, o) \leftarrow R_F(f) R_o(s) + \sum_{f' \in \mathcal{F}} \sum_{s' \in \mathcal{S}} T_F(f'|f, T_P(s')) T_o(s'|s) V_{k-1}(f', s')$
    $V_k(f, s) \leftarrow \max_{o \in \mathcal{O}} Q_k(f, s, o)$
  $\mu(f, s, o) = \arg\max_{o \in \mathcal{O}} Q(f, s, o)$
  return Options $\mathcal{O}$, metapolicy $\mu(f, s, o)$ and Q- and value functions $Q(f, s, o), V(f, s)$
end procedure

Associated with the low-level environment $\mathcal{E}$ is an LTL specification ϕ that can be decomposed into a liveness property $\phi_{liveness}$ and a safety property ϕ_{safety}. The set of propositions $\mathcal{P}$ is the union of the set of subgoal propositions $\mathcal{P}_G$ and the set of safety propositions $\mathcal{P}_S$. In our experiments, the safety property ϕ_{safety} defines a reward function R_S that penalizes violations of safety rules. ϕ_{safety} can be converted into an FSA $\mathcal{T}_S$, and the reward function can be defined on the states of $\mathcal{T}_S$. However, to limit the complexity of the formulation we assume that there is simply a reward (cost) associated with every safety proposition, and this reward function is $R_S : \mathcal{P}_S \rightarrow \mathbb{R}$. Therefore in the experiments, $R_{\mathcal{E}}$ is a combination of R_E, the environment reward function representing costs such as time, distance, or effort, and R_S, so that $R_{\mathcal{E}}(s, a) = R_E(s, a) + R_S(T_P(s))$.

$\phi_{liveness}$ is represented as a finite state automaton $\mathcal{T} = (\mathcal{F}, \mathcal{P}, T_F, R_F, f_0, f_g)$. $\mathcal{F}$ 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, $T_F : \mathcal{F} \times \mathcal{P} \times \mathcal{F} \rightarrow [0, 1]$. We assume a single initial state f_0 and final state f_g. There is also a reward function that assigns a reward to every state, $R_F : \mathcal{F} \rightarrow \mathbb{R}$.

From these parts we define a hierarchical SMDP $\mathcal{M} = (\mathcal{S} \times \mathcal{F}, \mathcal{A}, \mathcal{P}, \mathcal{O}, T_E \times T_P \times T_F, R_{SMDP}, \gamma)$. To reiterate, 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 option space (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. We also classify T_P, relating states to propositions, as a transition function because it helps to determine when FSA transitions occur. The reward function $R_{SMDP}(f, s, o) = R_F(f) R_o(s)$, so $R_F(s)$ is effectively a weighting on the option rewards. Lastly, the SMDP has the same discount factor γ as the environment $\mathcal{E}$.

Logical Value Iteration: A value function and Q-function are found for the SMDP using the Bellman update equations:

$$Q_k(f, s, o) \leftarrow R_F(f)R_o(s) + \sum_{f' \in \mathcal{F}} \sum_{s' \in \mathcal{S}} T_F(f'|f, T_P(s'))T_o(s'|s)V_{k-1}(f', s') \tag{3}$$

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

Eq. 3 differs from the generic equations for SMDP value iteration in that the transition function has an extra component, $\sum_{f' \in \mathcal{F}} T_F(f'|f, T_P(s'))$. This equation comes from Araki et al. (2019) and derives from the fact that, on every step in the environment, two transitions are applied: the option transition T_o, 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 Appendix A.1.

First, we describe the condition necessary 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 also 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 $-\infty < R_{\mathcal{E}}(s) < 0$ and if the episode terminates only 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 metapolicy $\mu^*(s)$ to always achieve the FSA goal state f_g. In our context, hierarchical optimality means that the metapolicy is optimal over the available options. Let $\mu'(f, s|f')$ be the hierarchically optimal goal-conditioned metapolicy for reaching FSA state f'. If the hierarchically optimal metapolicy equals the goal-conditioned metapolicy 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 $-\infty < R_{\mathcal{E}}(s) < 0$, 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 metapolicy, the LOF metapolicy will converge to the hierarchically optimal metapolicy.

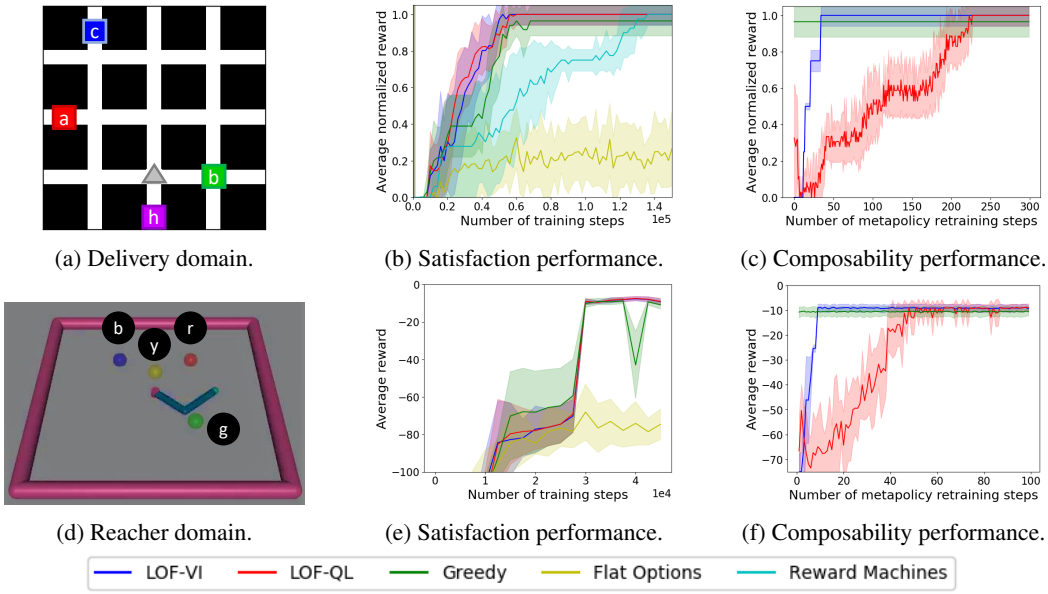
Consider the SMDP where planning is allowed over low-level actions, and let us call it the "hierarchical MDP" (HMDP) with optimal policy π^*_{HMDP}. We can then state the final theorem:

Theorem 3.1. *Given that the conditions for satisfaction and hierarchical optimality are met, the LOF hierarchically optimal metapolicy μ_g with optimal option subpolicies π_g is equivalent to the optimal policy π^*_{HMDP} and satisfies the task specification.*

4 Experiments & Results

Experiments: We performed experiments to demonstrate satisfaction and composability. For the satisfaction experiments, 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 metapolicy retraining steps it takes to learn an optimal metapolicy for a new task.

Environments: We measure the performance of LOF on two environments. The first environment is a discrete gridworld (Fig. 2a) called the "delivery domain," as it can represent a delivery truck



(a) Delivery domain. (b) Satisfaction performance. (c) Composability performance.

(d) Reacher domain. (e) Satisfaction performance. (f) Composability performance.

Figure 2: Performance on the satisfaction and composability experiments, averaged over all tasks. Note that `LOF-VI` composes new metapolicies in just 10-50 retraining steps. Results for the delivery domain are in the first row, for the reacher domain in the second row. All results are in Appendix A.5.

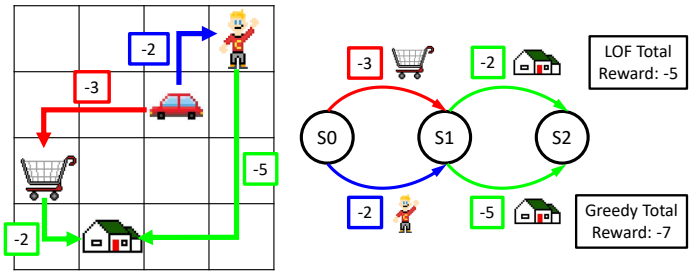


Figure 3: In this environment, the agent must either pick up the kid or go grocery shopping, and then go home. This is equivalent to the OR task. Starting at $S0$, the greedy algorithm picks the next step through the FSA with the lowest cost (in this case, picking up the kid), which leads to a higher overall cost. The LOF algorithm finds the optimal path through the FSA.

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. 2d). 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. In addition, both environments have a 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 $\Diamond(a \wedge \Diamond(b \wedge \Diamond(c \wedge \Diamond h))) \wedge \Box!o$ – "deliver package a, then b, then c, and then return home h. And always avoid obstacles." The next task is the "IF" task (equivalent to the task shown in Fig. 1b): $(\Diamond(c \wedge \Diamond a) \wedge \Box!can) \vee (\Diamond a \wedge \Diamond can) \wedge \Box!o$ – "deliver package c, and then a, unless a gets cancelled. And always avoid obstacles". We call the third task the "OR" task, $\Diamond((a \vee b) \wedge \Diamond c) \wedge \Box!o$ – "deliver package a or b, then c, and always avoid obstacles". The "composite" task has elements of all three of the previous tasks: $(\Diamond((a \vee b) \wedge \Diamond(c \wedge \Diamond h)) \wedge \Box!can) \vee (\Diamond((a \vee b) \wedge \Diamond h) \wedge \Diamond can) \wedge \Box!o$. "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 environment are equivalent, except that there are no obstacles for the reacher to avoid.

The sequential task is meant to show that planning is effective and efficient even for long-time horizon tasks. The "IF" task shows that the agent's policy can respond to unplanned changes, 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. 3. Lastly, the composite task shows that learning and planning are efficient and effective even for complex tasks.

Baselines: We test four baselines against our algorithm. We call our algorithm `LOF-VI`, short for "Logical Options Framework with Value Iteration," because it uses value iteration for its high-level planning. Our first baseline, `LOF-QL`, uses Q-learning instead. Unlike `LOF-VI`, `LOF-QL` does not need knowledge of T_F, the transition function of the FSA. `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. `Flat Options` uses the regular options framework with no knowledge of the FSA. The last baseline is `RM`, short for Reward Machines (Icarte et al., 2018). Whereas LOF learn options to accomplish subgoals, `RM` learns subpolicies for every FSA state.

Implementation: For the delivery domain, options were learned using Q-learning with an ϵ-greedy exploration policy. Options were learned simultaneously while switching the option used for exploration at every episode. `RM` was learned using the Q-learning for Reward Machines (QRM) algorithm described in (Icarte et al., 2018). For the reacher domain, 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 a 128×128 fully connected neural network. Deep-QRM was used to train `RM`. The implementation details are discussed more fully in Appendix A.2.

4.1 Results

Satisfaction: Results for the satisfaction experiments, averaged over all four tasks, are shown in Figs. 2b and 2e. (Results on all tasks are in Appendix A.5). 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 reacher domain 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 reacher domain, `RM` takes an order of magnitude more steps to train, so we left it out of the figure for clarity (see Appendix Fig. 10). However, in the reacher domain, `RM` eventually achieves a higher return than the LOF-based methods. This is because for the reacher domain, 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 metapolicy 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`. The results are shown in Figs. 2c and 2f. `Greedy` requires no retraining steps to "learn" a metapolicy 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 metapolicy 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.

Not Composable: The previous work most similar to ours is Icarte et al. (2018; 2019), which introduces a method to solve tasks defined by automata called Reward Machines. Their method learns a subpolicy for every state of the automaton; by transferring rewards between automaton states, they can achieve satisfaction and optimality. However, the learned policies have limited

composability because they are specific to the automaton; by contrast, LOF learns a subpolicy for every subgoal, independent of the automaton, and therefore the subpolicies 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 can 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). While these algorithms can guarantee satisfaction and, under certain conditions, optimality, they also have limited composability. 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). However, the meta-controller of Gordon et al. (2019) is not composable as it is 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.

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 HRL algorithms can travel much longer distances than a goal-oriented policy can travel on its own. Although these algorithms are "composable" in that they can navigate to arbitrary goals without further training, they are not able to solve tasks, and therefore none of them are capable of satisfying task specifications. Andreas et al. (2017) presents an algorithm for solving simple task "sketches" which is also composable; however, sketches are considerably less expressive than automata and linear temporal logic, which we use.

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 the policy is locally 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. 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.

6 Discussion and Conclusion

In this work we claim that the Logical Options Framework 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 highlighting 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 exactly 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 always 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 accomplish new tasks in very few iterations – about 10-50. Although `Greedy` requires no retraining steps, it is a tiny fraction of the tens of thousands of steps required to learn the original policy. Lastly, we have shown that LOF can learn policies efficiently, and that it can be used with a variety of domains and policy-learning algorithms. In fact, any policy-learning algorithm where it is possible to extract a value (expected cumulative reward) for the policy's execution on the subgoals is fully compatible with LOF, including value iteration and PPO.

Thus, we have proven and demonstrated LOF's features of satisfaction, optimality, and composability, while also reviewing the compromises involved in achieving this goal. We hope that this framework can be useful in practical settings that are governed by complex and changeable rules.

REFERENCES

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

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

Brandon Araki, Kiran Vodrahalli, Thomas Leech, Cristian Ioan Vasile, Mark Donahue, and Daniela Rus. Learning to plan with logical automata. In *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, June 2019. doi: 10.15607/RSS.2019.XV.064.

Brandon Araki, Kiran Vodrahalli, Thomas Leech, Cristian Ioan Vasile, Mark Donahue, and Daniela Rus. Deep bayesian nonparametric learning of rules and plans from demonstrations with a learned automaton prior. In *AAAI*, pp. 10026–10034, 2020.

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

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

Alberto Camacho, Rodrigo Toro Icarte, Toryn Q Klassen, Richard Anthony Valenzano, and Sheila A McIlraith. Ltl and beyond: Formal languages for reward function specification in reinforcement learning. In *IJCAI*, volume 19, pp. 6065–6073, 2019.

Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 2001. ISBN 978-0-262-03270-4.

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

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

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

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

Aleksandra Faust, Kenneth Oslund, Oscar Ramirez, Anthony Francis, Lydia Tapia, Marek Fiser, and James Davidson. 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.

Dibya Ghosh, Abhishek Gupta, and Sergey Levine. Learning actionable representations with goal-conditioned policies. *arXiv preprint arXiv:1811.07819*, 2018.

Nakul Gopalan, Michael L Littman, James MacGlashan, Shawn Squire, Stefanie Tellex, John Winder, Lawson LS Wong, et al. Planning with abstract markov decision processes. In *Twenty-Seventh International Conference on Automated Planning and Scheduling*, 2017.

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

Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-constrained reinforcement learning. *arXiv preprint arXiv:1801.08099*, 2018.

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*, pp. 2107–2116, 2018.

Rodrigo Toro Icarte, Ethan Waldie, Toryn Klassen, Rick Valenzano, Margarita Castro, and Sheila McIlraith. Learning reward machines for partially observable reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 15523–15534, 2019.

León Illanes, Xi Yan, Rodrigo Toro Icarte, and Sheila A McIlraith. 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.

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

Yen-Ling Kuo, Boris Katz, and Andrei Barbu. Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of ltl formulas. *arXiv preprint arXiv:2006.01110*, 2020.

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

Xiao Li, Zachary Serlin, Guang Yang, and Calin Belta. A formal methods approach to interpretable reinforcement learning for robotic planning. *Science Robotics*, 4(37), 2019.

Gary Marcus. The next decade in ai: four steps towards robust artificial intelligence. *arXiv preprint arXiv:2002.06177*, 2020.

George Rupert Mason, Radu Constantin Calinescu, Daniel Kudenko, and Alec Banks. Assured reinforcement learning with formally verified abstract policies. In *9th International Conference on Agents and Artificial Intelligence (ICAART)*. York, 2017.

Yoonseon Oh, Roma Patel, Thao Nguyen, Baichuan Huang, Ellie Pavlick, and Stefanie Tellex. Planning with state abstractions for non-markovian task specifications. *arXiv preprint arXiv:1905.12096*, 2019.

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

Chris Paxton, Vasumathi Raman, Gregory D Hager, and Marin Kobilarov. 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.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

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

Lim Zun Yuan, Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Modular deep reinforcement learning with temporal logic specifications. *arXiv preprint arXiv:1909.11591*, 2019.

A Appendix

A.1 Proofs and Conditions for Satisfaction and Optimality

Definition A.1. *Let the reward function of the environment be $R_{\mathcal{E}}(s, a)$. Let $V^{\pi'}(s|s')$ be the optimal cumulative reward for reaching s' from s, where π' is the optimal goal-conditioned policy for achieving s'. Let s_g be the state associated with the subgoal, and let π_g be the optimal goal-conditioned policy associated with reaching s_g. Let π^* be the optimal policy for the environment $\mathcal{E}$.*

Condition A.1. *The optimal policy for the option must be the same as the goal-conditioned policy that has subgoal s_g as its goal: $\pi^*(s) = \pi_g(s|s_g)$. In other words, $V^{\pi_g}(s|s_g) > V^{\pi'}(s|s') \;\; \forall s, s' \neq s_g$.*

This condition guarantees that the optimal option policy will always reach the subgoal s_g. It can be achieved by setting all rewards $-\infty < R_{\mathcal{E}}(s, a) < 0$ and terminating the episode only when the agent reaches s_g. Therefore the expected return for reaching s_g is a bounded negative number, and the expected return for all other states is $-\infty$.

Lemma A.2. *Given that there is an option for every subgoal and that all the options meet Condition A.1, there exists a metapolicy that can reach the FSA goal state from any non-trap state in the FSA.*

Proof. This follows from the fact that transitions in the FSA are determined by achieving subgoals, and it is given that there exists an option for achieving every subgoal. Therefore, it is possible for the agent to execute any sequence of subgoals, and at least one of those sequences must satisfy the task specification since the FSA representing the task specification is finite and satisfiable. □

Definition A.2. *From Dietterich (2000): A **hierarchically optimal** policy for an MDP or SMDP is a policy that achieves the highest cumulative reward among all policies consistent with the given hierarchy.*

In our case, this means that the hierarchically optimal metapolicy is optimal over the available options.

Definition A.3. *Let the cumulative reward function of an option o started at state s be $R_o(s)$. Let the reward function on the SMDP be $R_{SMDP}(f, s, o) = R_F(f)R_o(s)$ with $R_F(f) \geq 0$[1]. Let $V^{\mu'}(f, s|f')$ be the hierarchically optimal return for reaching f' from (f, s), where μ' is the hierarchically optimal goal-conditioned policy for achieving f'. Let μ^* be the hierarchically optimal policy for the SMDP. Let f_g be the goal state, and μ_g be the hierarchically optimal goal-conditioned policy for achieving the goal state.*

Condition A.3. *The hierarchically optimal metapolicy must be the same as the goal-conditioned metapolicy that has the FSA goal state f_g as its goal: $\mu^*(f, s) = \mu_g(f, s|f_g)$. In other words, $V^{\mu_g}(f, s|f_g) > V^{\mu'}(f, s|f') \;\; \forall f, s, f' \neq f_g$.*

This condition guarantees that the hierarchically optimal metapolicy will always go to the FSA goal state f_g (thereby satisfying the specification). Here is an example of how this condition can be achieved: If $-\infty < R_{\mathcal{E}}(s, a) < 0 \;\; \forall s$, then $R_o(s) < 0 \;\; \forall o, s$. Then if $R_F(f) > 0$ (in our experiments, we set $R_F(f) = 1 \;\; \forall f$), $R_{SMDP}(f, s, o) = R_F(f)R_o(s) < 0$, and if the episode only terminates when the agent reaches the goal state, then the expected return for reaching f_g is a bounded negative number, and the expected return for all other states is $-\infty$.

Lemma A.4. *From (Sutton et al., 1999): Value iteration on an SMDP converges to the hierarchically optimal policy.*

Therefore, the metapolicy found using the Logical Options Framework converges to a hierarchically optimal metapolicy that satisfies the task specification as long as Conditions A.1 and A.3 are met.

Definition A.4. *Consider the SMDP where planning is allowed over the low-level actions instead of the options. We will call this the hierarchical MDP (HMDP), as this MDP is the product of the low-level environment $\mathcal{E}$ and the logical FSA $\mathcal{T}$. Let $R_F(f) > 0 \; \forall f$, and let $R_{HMDP}(f, s, a) = R_F(f)R_{\mathcal{E}}(s, a)$, and let π^*_{HMDP} be the optimal policy for the HMDP.*

Theorem A.5. *Given Conditions A.1 and A.3, the hierarchically optimal metapolicy μ_g with optimal option policies π_g is equivalent to the HMDP optimal policy π^* and satisfies the task specification.*

Proof. By Condition A.1, every subgoal has an option associated with it whose optimal policy is to go to the subgoal. By Condition A.3, the hierarchically optimal metapolicy will reach the FSA

[1]The assumption that $R_{SMDP}(f, s, o) = R_F(f)R_o(s)$ and $R_{HMDP}(f, s, a) = R_F(f)R_{\mathcal{E}}(s, a)$ can be relaxed so that R_{SMDP} and R_{HMDP} are functions that are monotonic increasing in the low-level rewards R_o and $R_{\mathcal{E}}$, respectively.

goal state f_g. The metapolicy can only accomplish this by going to the subgoals in a sequence that satisfies the task specification. It does this by executing a sequence of options corresponding to a satisfying sequence of subgoals. Therefore, since $R_F(f) > 0 \ \forall f$ and $R_{SMDP}(f, s, o) = R_F(f)R_o(s)$, the cumulative reward is a positive linear combination of the option rewards, and since all option rewards are optimal with respect to the environment and the metapolicy is optimal over the options, our algorithm attains the optimal reward. □

A.2 Experimental Implementation

We discuss the implementation details of the experiments in this section. Because the delivery and reacher domains are analogous, we discuss the delivery domain first in every section and then briefly relate how the same formulation applies to the reacher domain as well.

A.2.1 Propositions

The delivery domain has 7 propositions plus 4 composite propositions. The subgoal propositions are $\mathcal{P}_G = \{a, b, c, h\}$. Each of these propositions is associated with a single state in the environment (see Fig. 8a). The safety propositions are $\mathcal{P}_S = \{can, o, e\}$. o is the obstacle proposition. It is associated with many states – the black squares in Fig. 8a. e is the empty proposition, associated with all of the white squares in the domain. This is the default proposition for when there are no other active propositions. can is the "cancelled" proposition, representing when one of the subgoals has been cancelled. Technically propositions must be associated with a state via the labelling function $T_P : \mathcal{S} \times \mathcal{P} \to [0, 1]$. However, because we want this proposition to be triggered at a certain time and place in the experiments, we manually toggle the `True/False` value of the proposition based on the location of the agent and FSA state of the environment.

One of the framework's assumptions is that multiple propositions cannot be true at the same state. However, it is reasonable for can to be true at the subgoals, and therefore we introduce 4 composite propositions, $ca = a \wedge can$, $cb = b \wedge can$, $cc = c \wedge can$, $ch = h \wedge can$. These can be counted as safety propositions without affecting the operation of the algorithm.

The reacher domain has analogous propositions. The subgoals are r, g, b, y and correspond to a, b, c, h. The environment does not contain obstacles o but does have can and e, and it also has the composite propositions cr, cg, cb, cy for when can is true at the same time that a subgoal proposition is true. Another difference is that the subgoal propositions are associated with a small spherical region instead of a single state as in the delivery domain; this is a necessity for continuous domains and unfortunately breaks one of our conditions for optimality because the subgoals are now associated with multiple states instead of a single state. However, the LOF metapolicy will still converge to a hierarchically optimal policy.

A.3 Reward Functions

Next, we define the reward functions of the physical environment R_E, safety propositions R_S, and FSA states R_F. We realize that often in reinforcement learning, the algorithm designer has no control over the reward functions of the environment. However, in our case, there are no publicly available environments such as OpenAI Gym or the DeepMind Control Suite that we know of that have a high-level FSA built-in. Therefore, anyone implementing our algorithm will likely have to implement their own high-level FSA and define the rewards associated with it.

Our low-level environment reward function $R_E : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is defined to be $-1 \ \forall s, a$. In other words, it is a time/distance cost.

We assign costs to the safety propositions by defining the reward function $R_S : \mathcal{P}_S \to \mathbb{R}$. All of the costs are 0 except for the obstacle cost, $R_S(o) = -1000$. Therefore, there is a very high penalty for encountering an obstacle.

We define the environment reward function $R_{\mathcal{E}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ to be $R_{\mathcal{E}}(s, a) = R_E(s, a) + R_S(T_P(s))$. In other words, it is the sum of R_E and R_S. This reward function meets Condition A.1 for the optimal option policies to always converge to their subgoals.

Lastly, we define $R_F : \mathcal{F} \to \mathbb{R}$ to be $R_F(f) = 1 \; \forall f$. Therefore the SMDP cost $R_S MDP(f, s, o) = R_o(s)$ and meets Condition A.3 so that the LOF metapolicy converges to the optimal policy.

The reacher environment has analogous reward functions. The safety reward function $R_S(p) = 0 \; \forall p \in \mathcal{P}_S$ because there is no obstacle proposition. Also, the physical environment reward function differs during option training and metapolicy learning. For metapolicy learning, the reward function is $R_E(s, a) = -a^\top a - 0.1$ – a time cost and an actuation cost. During option training, we speed learning by adding the distance to the goal state as a cost, instead of a time cost: $R_E(s, a) = -a^\top a - ||s - s_g||^2$. Although the reward functions and value functions are different, the costs are analogous and lead to good performance as seen in the results. Note that this method can't be used for Reward Machines, because it trains subpolicies for FSA states, and the subgoals for FSA states are not known ahead of time, so distance to subgoal cannot be calculated.

A.4 TASKS

We test the environments on four tasks, a "sequential" task (Fig. 4), an "IF" task (Fig. 5), an "OR" task (Fig. 6), and a "composite" task (Fig. 7). The reacher domain has the same tasks, expect r, g, b, y replace a, b, c, h, and there are no obstacles o. Note that in the LTL formulae, $\Box!o$ is the safety property ϕ_{safety}; the preceding part of the formula is the liveness property $\phi_{liveness}$ used to construct the FSA.

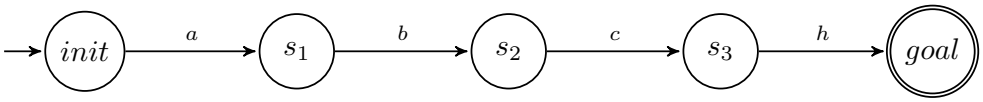


Figure 4: FSA for the sequential task. The LTL formula is $\Diamond(a \wedge \Diamond(b \wedge \Diamond(c \wedge \Diamond h))) \wedge \Box!o$. The natural language interpretation is "Deliver package a, then b, then c, and then return home h. And always avoid obstacles o".

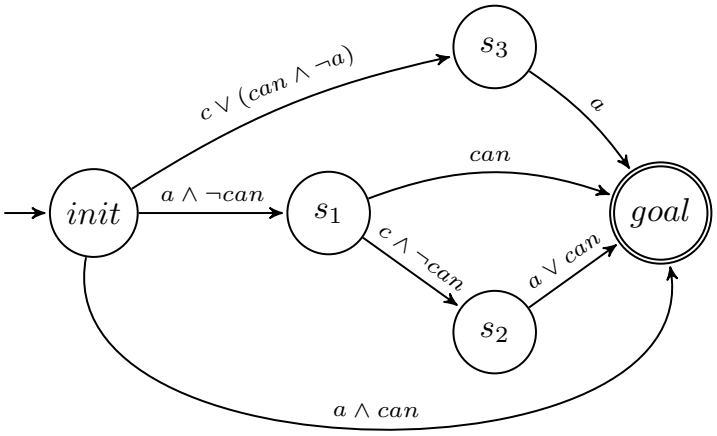


Figure 5: FSA for the IF task. The LTL formula is $(\Diamond(c \wedge \Diamond a) \wedge \Box!can) \vee (\Diamond a \wedge \Diamond can) \wedge \Box!o$. The natural language interpretation is "Deliver package c, and then a, unless a gets cancelled. And always avoid obstacles o".

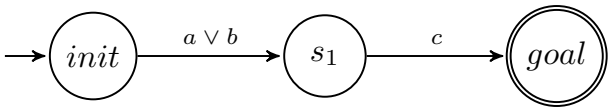


Figure 6: FSA for the OR task. The LTL formula is $\Diamond((a \vee b) \wedge \Diamond c) \wedge \Box!o$. The natural language interpretation is "Deliver package a or b, then c, and always avoid obstacles o".

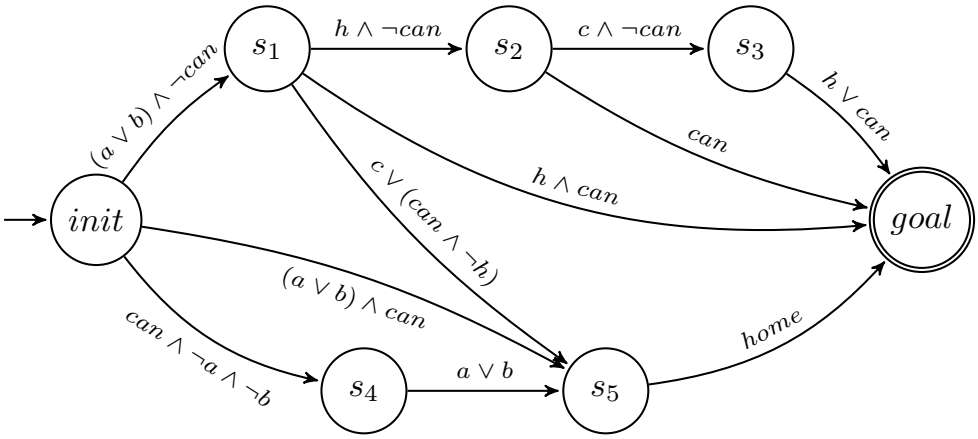


Figure 7: FSA for the composite task. The LTL formula is $(\Diamond((a \vee b) \wedge \Diamond(c \wedge \Diamond h)) \wedge \Box!can) \vee (\Diamond((a \vee b) \wedge \Diamond h) \wedge \Diamond can) \wedge \Box!o$. The natural language interpretation is "Deliver package a or b, and then c, unless c gets cancelled, and then return to home h. And always avoid obstacles".

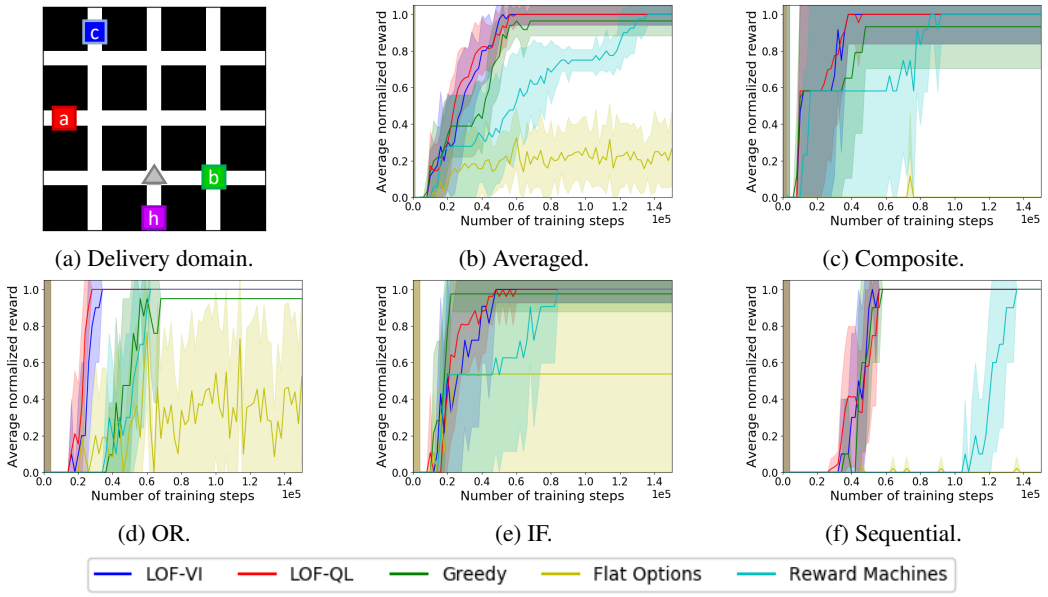
A.5 Full Experimental Results

For the satisfaction experiments for the delivery domain, 10 policies were trained for each task and for each baseline. Training was done for 1600 episodes, with 100 steps per episode. Every 2000 training steps, the policies were tested on the domain and the returns recorded. For this discrete domain, we know the minimum and maximum possible returns for each task, and we normalized the returns using these minimum and maximum returns. The error bars are the standard deviation of the returns over the 10 policies' rollouts.

For the satisfaction experiments for the reacher domain, a single policy was trained for each task and for each baseline. The baselines were trained for 900 epochs, with 50 steps per epoch. Every 2500 training steps, each policy was tested by doing 10 rollouts and recording the returns. For the `RM` baseline, training was for 1000 epochs with 800 steps per epoch, and the policy was tested every 8000 training steps. Because we don't know the minimum and maximum rewards for each task, we did not normalize the returns. The error bars are the standard deviation over the 10 rollouts for each baseline.

For the composability experiments, a set of options was trained once, and then metapolicing training using `LOF-VI`, `LOF-QL`, and `Greedy` was done for each task. Returns were recorded at every training step by rolling out each baseline 10 times. The error bars are the standard deviations on the 10 rollouts.

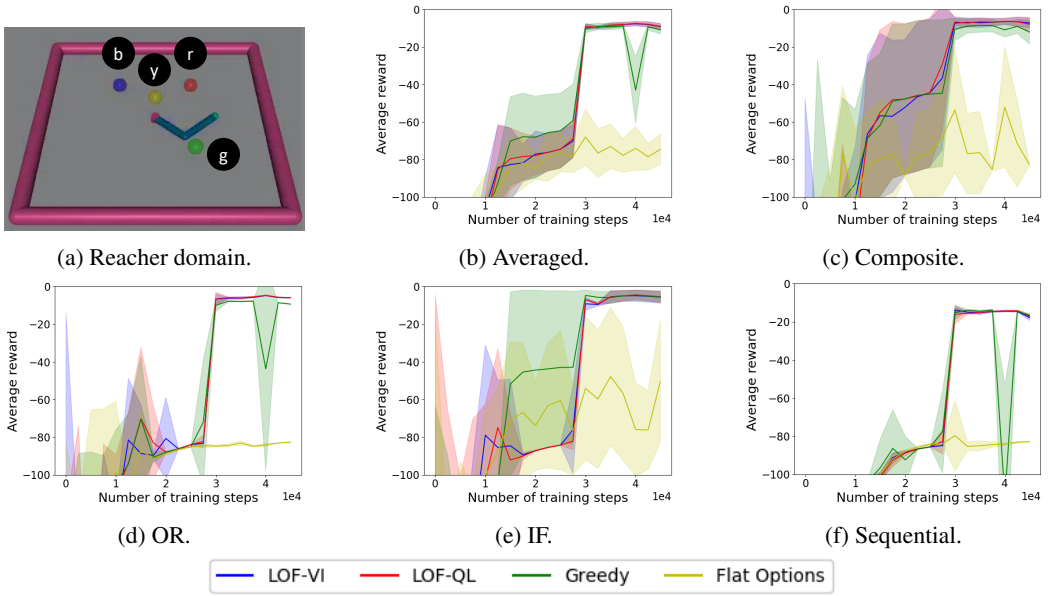
Code and videos of the domains and tasks are in the supplement.



(a) Delivery domain. (b) Averaged. (c) Composite.

(d) OR. (e) IF. (f) Sequential.

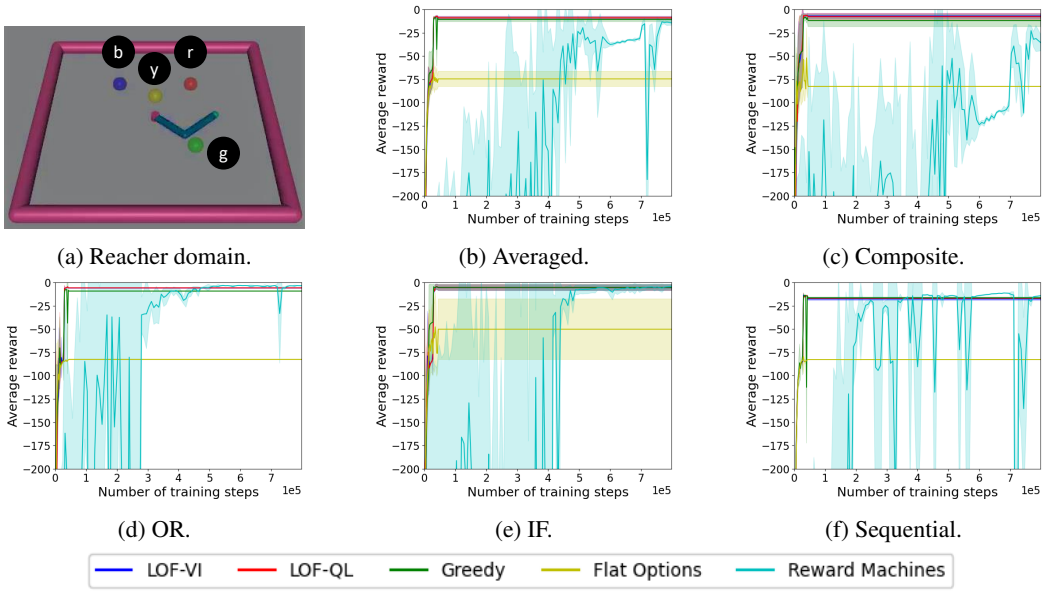
Figure 8: All satisfaction experiments on the delivery domain. Notice how for the composite and OR tasks (Figs. 8c and 8d), the `Greedy` baseline plateaus before `LOF-VI` and `LOF-QL`. This is because `Greedy` chooses a suboptimal path through the FSA, whereas `LOF-VI` and `LOF-QL` find an optimal path. Also, notice that `RM` takes many more training steps to achieve the optimal cumulative reward. This is because for `RM`, the only reward signal is from reaching the goal state. It takes a long time for the agent to learn an optimal policy from such a sparse reward signal. This is particularly evident for the sequential task (Fig. 8f), which requires the agent to take a longer sequence of actions/FSA states before reaching the goal. The options-based algorithms train much faster because when training the options, the agent receives a reward for reaching each subgoal, and therefore the reward signal is much richer.



(a) Reacher domain. (b) Averaged. (c) Composite.

(d) OR. (e) IF. (f) Sequential.

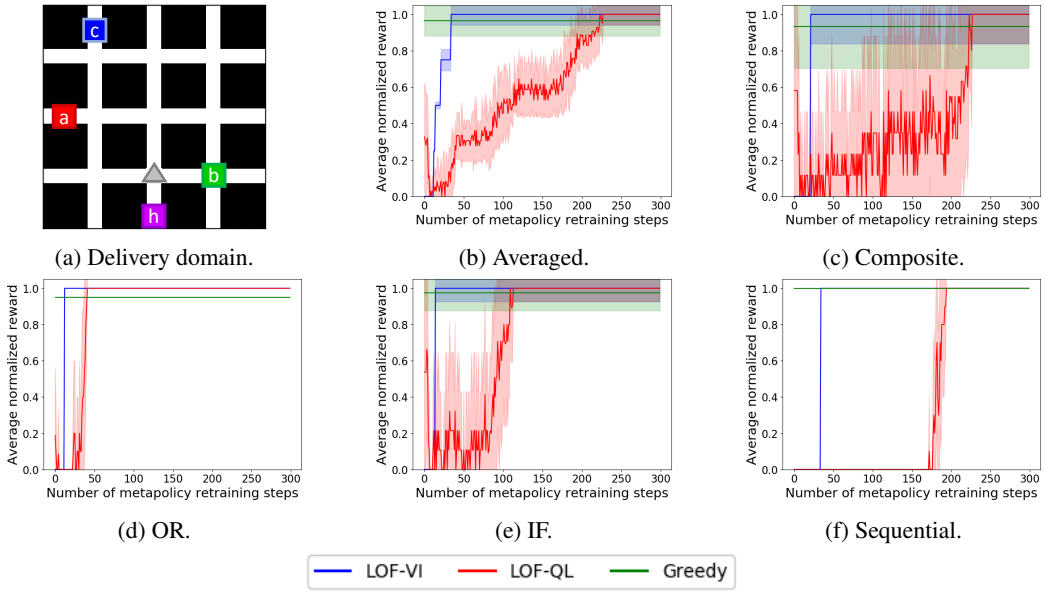
Figure 9: Satisfaction experiments for the reacher domain, without `RM` results. The results are equivalent to the results on the delivery domain.



(a) Reacher domain. (b) Averaged. (c) Composite.

(d) OR. (e) IF. (f) Sequential.

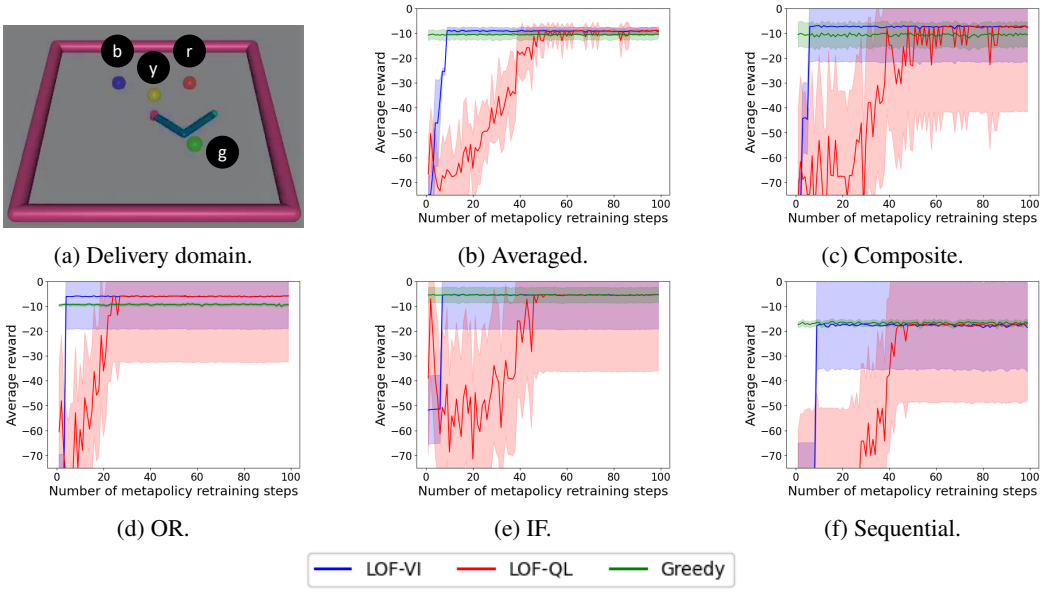
Figure 10: Satisfaction experiments for the reach domain, including RM results. RM takes significantly more training steps to train than the other baselines, although it eventually reaches and surpasses the cumulative reward of the other baselines. This is because for the continuous domain, we violate some of the conditions required for optimality when using the Logical Options Framework – in particular, the condition that each subgoal is associated with a single state. In a continuous environment, this condition is impossible to meet, and therefore we made the subgoals small spherical regions, and we only made the subgoals associated with specific Cartesian coordinates and not velocities (which are also in the state space). Meanwhile, the optimality conditions of RM are looser and were not violated, which is why it achieves a higher final cumulative reward.



(a) Delivery domain. (b) Averaged. (c) Composite.

(d) OR. (e) IF. (f) Sequential.

Figure 11: All composability experiments for the delivery domain.



(a) Delivery domain. (b) Averaged. (c) Composite.

(d) OR. (e) IF. (f) Sequential.

Figure 12: All composability experiments for the reacher domain.