A. Formulation of Logical Options Framework with Safety Automaton

In this section, we present a more general formulation of LOF than that presented in the paper. In the paper, we make two assumptions that simplify the formulation. The first assumption is that the LTL specification can be divided into two independent formulae, a liveness property and a safety property: $\phi = \phi_{liveness} \wedge \phi_{safety}$. However, not all LTL formulae can be factored in this way. We show how LOF can be applied to LTL formulae that break this assumption. The second assumption is that the safety property takes a simple form that can be represented as a penalty on safety propositions. We show how LOF can be used with arbitrary safety properties.

A.1. Automata and Propositions

All 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), so that automaton $\mathcal{W} = \mathcal{W}_{liveness} \times \mathcal{W}_{safety}$. This is a generalization of the assumption that all LTL formulae can be divided into liveness and safety properties $\phi_{liveness}$ and ϕ_{safety}. The liveness property $\mathcal{W}_{liveness}$ must be an FSA, although this assumption could also be loosened to allow it to be a deterministic Büchi automaton via some minor modifications (allowing multiple goal states to exist and continuing episodes indefinitely, even once a goal state has been reached).

As in the main text, we assume that there are three types of propositions – subgoals $\mathcal{P}_G$, safety propositions $\mathcal{P}_S$, and event propositions $\mathcal{P}_E$. The event propositions have set values and can occur in both $\mathcal{W}_{liveness}$ and $\mathcal{W}_{safety}$. Safety propositions only appear in $\mathcal{W}_{safety}$. Subgoal propositions only appear in $\mathcal{W}_{liveness}$. Each subgoal may only be associated with one state. Note that after writing a specification and decomposing it into $\mathcal{W}_{liveness}$ and $\mathcal{W}_{safety}$, it is possible that some subgoals may unexpectedly appear in $\mathcal{W}_{safety}$. This can be dealt with by creating "safety twins" of each subgoal – safety propositions that are associated with the same low-level states as the subgoals and can therefore substitute for them in $\mathcal{W}_{safety}$.

Subgoals are propositions that the agent must achieve in order to reach the goal state of $\mathcal{W}_{liveness}$. Although event propositions can also define transitions in $\mathcal{W}_{liveness}$, we assume that "achieving" them is not necessary in order to reach the goal state. In other words, we assume that from any state in $\mathcal{W}_{liveness}$, there is a path to the goal state that involves only subgoals. This is because in our formulation, the event propositions are meant to serve as propositions that the agent has no control over, such as receiving a phone call. If satisfaction of the liveness property were to depend on such a proposition, then it would be impossible to guarantee satisfaction. However, if the user is unconcerned with guaranteeing satisfaction, then specifying a liveness property in which satisfaction depends on event propositions is compatible with LOF.

Safety propositions may only occur in $\mathcal{W}_{safety}$ and are associated with things that the agent "must avoid". This is because every state of $\mathcal{W}_{safety}$ is an accepting state (Alpern & Schneider, 1987), so all transitions between the states are non-violating. However, any undefined transition is not allowed and is a violation of the safety property. In our formulation, we assign costs to violations, so that violations are allowed but come at a cost. In practice, it also may be the case that the agent is in a low-level state from which it is impossible to reach the goal state without violating the safety property. In our formulation, satisfaction of the liveness property (but not the safety property) is still guaranteed in this case, as the finite cost associated with violating the rule is less than the infinite cost of not satisfying the liveness property, so the optimal policy for the agent will be to violate the rule in order to satisfy the task (see the proofs, Appendix B). This scenario can be avoided in several ways. For example, do not specify an environment in which it is only possible for the agent to satisfy the task by violating a rule. Or, instead of prioritizing satisfaction of the task, it is possible to instead prioritize satisfaction of the safety property. In this case, satisfaction of the liveness property would not be guaranteed but satisfaction of the safety property would be guaranteed. This could be accomplished by terminating the rollout if a safety violation occurs.

We assume that event propositions are observed – in other words, that we know the values of the event propositions from the start of a rollout. This is because we are planning in a fully observable setting, so we must make this assumption to guarantee convergence to an optimal policy. However, the partially observable case is much more interesting, in which the values of the event propositions are not known until the agent checks or the environment randomly reveals their values. This case is beyond the scope of this paper; however, LOF can still guarantee satisfaction and composability in this setting, just not optimality.

Proposition labeling functions relate states to propositions: $T_{P_G} : \mathcal{S} \to 2^{\mathcal{P}_G}$, $T_{P_S} : \mathcal{S} \to 2^{\mathcal{P}_S}$, and $T_{P_E} : 2^{\mathcal{P}_E} \to \{0, 1\}$.

Given these definitions of propositions, it is possible to define the liveness and safety properties formally. $\mathcal{W}_{liveness} = (\mathcal{F}, \mathcal{P}_G \cup \mathcal{P}_E, T_F, R_F, f_0, f_g)$. $\mathcal{F}$ is the set of states of the liveness property. The propositions can be either subgoals $\mathcal{P}_G$ or event propositions $\mathcal{P}_E$. The transition function relates the current FSA state and active propositions to the next FSA state, $T_F : \mathcal{F} \times 2^{\mathcal{P}_G} \times 2^{\mathcal{P}_E} \times \mathcal{F} \to [0, 1]$.

The reward function assigns a reward to the current FSA state, $R_F : \mathcal{F} \to \mathbb{R}$. We assume there is one initial state f_0 and one goal state f_g.

The safety property is a Büchi automaton $\mathcal{W}_{safety} = (\mathcal{F}_S, \mathcal{P}_S \cup \mathcal{P}_E, T_S, R_S, F_0)$. $\mathcal{F}_S$ are the states of the automaton. The propositions can be safety propositions $\mathcal{P}_S$ or event propositions $\mathcal{P}_E$. The transition function T_S relates the current state and active propositions to the next state, $T_S : \mathcal{F}_S \times 2^{\mathcal{P}_S} \times 2^{\mathcal{P}_E} \times \mathcal{F}_S \to [0, 1]$. The reward function relates the automaton state and safety propositions to rewards (or costs), $R_S : \mathcal{F}_S \times 2^{\mathcal{P}_S} \to \mathbb{R}$. F_0 defines the set of initial states. We do not specify an accepting condition because for safety properties, every state is an accepting state.

A.2. The Environment MDP

There is a low-level environment MDP $\mathcal{E} = (\mathcal{S}, \mathcal{A}, R_E, T_E, \gamma)$. $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space. They can be either discrete or continuous. R_E is the low-level reward function that characterizes, for example, time, distance, or actuation costs. $T_E : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition function and γ is the discount factor. Unlike in the simpler formulation in the paper, we do not combine R_E and the safety automaton reward function R_S in the MDP formulation $\mathcal{E}$.

A.3. Logical Options

We associate every subgoal p_g with an option $o_{p_g} = (\mathcal{I}_{p_g}, \pi_{p_g}, \beta_{p_g}, R_{p_g}, T_{p_g})$. Every o_{p_g} has a policy π_{p_g} whose goal is to reach the state s_{p_g} where p_g is true. Option policies are learned by training on the product of the environment and the safety automaton, $\mathcal{E} \times \mathcal{W}_{safety}$ and terminating training only when s_{p_g} is reached. $R_{\mathcal{E}} : \mathcal{F}_S \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function of the product MDP $\mathcal{E} \times \mathcal{W}_{safety}$. There are many reward-shaping policy-learning algorithms that specify how to define $R_{\mathcal{E}}$. In fact, learning a policy for $\mathcal{E} \times \mathcal{W}_{safety}$ is the sort of hierarchical learning problem that many reward-shaping algorithms excel at, including Reward Machines (Icarte et al., 2018) and (Li et al., 2017). This is because in LOF, safety properties are not composable, so using a learning algorithm that is satisfying and optimal but not composable to learn the safety property is appropriate. Alternatively, there are many scenarios where $\mathcal{W}_{safety}$ is a trivial automaton in which each safety proposition is associated with its own state, as we describe in the main paper, so penalties can be assigned to propositions and the state of the agent in $\mathcal{W}_{safety}$ can be ignored.

Note that since the options are trained independently, one limitation of our formulation is that the safety properties cannot depend on the liveness state. In other words, when an agent reaches a new subgoal, the safety property cannot change. However, the workaround for this is not too complicated. First, if the liveness state affects the safety property, this implies that liveness propositions such as subgoals may be in the safety property. In this case, as we described above, the subgoals present in the safety property need to be substituted with "safety twin" propositions. Then during option training, a policy-learning algorithm must be chosen that will learn sub-policies for all of the safety property states,

Algorithm 2 Learning and Planning with Logical Options

Given:
Propositions $\mathcal{P}$ partitioned into subgoals $\mathcal{P}_G$, safety propositions $\mathcal{P}_S$, and event propositions $\mathcal{P}_E$
$\mathcal{W}_{liveness} = (\mathcal{F}, \mathcal{P}_G \cup \mathcal{P}_E, T_F, R_F, f_0, f_g)$
$\mathcal{W}_{safety} = (\mathcal{F}_S, \mathcal{P}_S \cup \mathcal{P}_E, T_S, R_S, F_0)$
Low-level MDP $\mathcal{E} = (\mathcal{S}, \mathcal{A}, R_E, T_E, \gamma)$
Proposition labeling functions $T_{P_G} : \mathcal{S} \to 2^{\mathcal{P}_G}$, $T_{P_S} : \mathcal{S} \to 2^{\mathcal{P}_S}$, and $T_{P_E} : 2^{\mathcal{P}_E} \to \{0, 1\}$
To learn:
Set of options $\mathcal{O}$, one for each subgoal proposition $p \in \mathcal{P}_G$
Meta-policy $\mu(f, f_s, s, o)$ along with $Q(f, f_s, s, o)$ and $V(f, f_s, s)$
Learn logical options:
For every p in $\mathcal{P}_G$, learn an option for achieving p, $o_p = (\mathcal{I}_{o_p}, \pi_{o_p}, \beta_{o_p}, R_{o_p}, T_{o_p})$
$\mathcal{I}_{o_p} = \mathcal{S}$
$\beta_{o_p} = \begin{cases} 1 & \text{if } p \in T_{P_G}(s) \\ 0 & \text{otherwise} \end{cases}$
π_{o_p} = optimal policy on $\mathcal{E} \times \mathcal{W}_{safety}$ with rollouts terminating when $p \in T_{P_G}(s)$
$T_{o_p}(f'_s, s' | f_s, s) = \begin{cases} \sum_{k=1}^{\infty} p(f'_s, k)\gamma^k & \text{if } p \in T_P(s') \\ 0 & \text{otherwise} \end{cases}$
$R_{o_p}(f_s, s) = \mathbb{E}[\mathcal{R}_{\mathcal{E}}(f_s, s, a_1) + \gamma \mathcal{R}_{\mathcal{E}}(f_{s,1}, s_1, a_2) + \cdots + \gamma^{k-1} \mathcal{R}_{\mathcal{E}}(f_{s,k-1}, s_{k-1}, a_k)]$
Find a meta-policy μ over the options:
Initialize $Q : \mathcal{F} \times \mathcal{F}_S \times \mathcal{S} \times \mathcal{O} \to \mathbb{R}$ and $V : \mathcal{F} \times \mathcal{F}_S \times \mathcal{S} \to \mathbb{R}$ to 0
for $(k, f, f_s, s) \in [1, \ldots, n] \times \mathcal{F} \times \mathcal{F}_S \times \mathcal{S}$: **do**
  for $o \in \mathcal{O}$: **do**
    $Q_k(f, f_s, s, o) \leftarrow R_F(f) R_o(f_s, s) + \sum_{f' \in \mathcal{F}} \sum_{f'_s \in \mathcal{F}_S} \sum_{\bar{p}_e \in 2^{\mathcal{P}_E}} \sum_{s' \in \mathcal{S}} T_F(f' | f, T_{P_G}(s'), \bar{p}_e) T_S(f'_s | f_s, T_{P_S}(s'), \bar{p}_e) T_{P_E}(\bar{p}_e) T_o(s' | s) V_{k-1}(f', f'_s, s')$
  end for
  $V_k(f, f_s, s) \leftarrow \max_{o \in \mathcal{O}} Q_k(f, f_s, s, o)$
end for
$\mu(f, f_s, s, o) = \arg\max_{o \in \mathcal{O}} Q(f, f_s, s, o)$
Return: Options $\mathcal{O}$, meta-policy $\mu(f, f_s, s, o)$, and $Q(f, f_s, s, o), V(f, f_s, s)$

even if those states are only reached after completing a complicated task (for example, all of the sub-policies could be trained in parallel as in (Icarte et al., 2018)). Lastly, during meta-policy learning and during rollouts, when a new option is chosen, the current state of the safety property must be passed to the new option.

The components of the logical options are defined starting at Alg. 2 line 5. Note that for stochastic low-level transitions, the number of time steps k at which the option terminates is stochastic and characterized by a distribution function. In general this distribution function must be learned, which is a challenging problem. However, there are many approaches to solving this problem; (Abel & Winder, 2019) contains an excellent discussion.

The most notable difference between the general formulation and the formulation in the paper is that the option policy, transition, and reward functions are functions of the safety automaton state f_s as well as the low-level state s. This makes Logical Value Iteration more complicated, because in the paper, we could assume we knew the final state of each option (i.e., the state of its associated subgoal s_g). But now, although we still assume that the option will terminate at s_g, we do not know which safety automaton state it will terminate in, so the transition model must learn a distribution over safety automaton states, and Logical Value Iteration must account for this uncertainty.

A.4. Hierarchical SMDP

Given a low-level environment $\mathcal{E}$, a liveness property $\mathcal{W}_{liveness}$, a safety property $\mathcal{W}_{safety}$, and logical options $\mathcal{O}$, we can define a hierarchical semi-Markov Decision Process (SMDP) $\mathcal{M} = \mathcal{E} \times \mathcal{W}_{liveness} \times \mathcal{W}_{safety}$ with options $\mathcal{O}$ and reward function R_{SMDP}. This SMDP differs significantly from the SMDP in the paper in that the safety property $\mathcal{W}_{safety}$ is now an integral part of the formulation. $R_{SMDP}(f, f_s, s, o) = R_F(f)R_o(f_s, o)$.

A.5. Logical Value Iteration

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

$$Q_k(f, f_s, s, o) \leftarrow R_F(f)R_o(f_s, s) + \sum_{f' \in \mathcal{F}} \sum_{f'_s \in \mathcal{F}_S} \sum_{\bar{p}_e \in 2^{\mathcal{P}_E}} \sum_{s' \in \mathcal{S}} T_F(f'|f, T_{P_G}(s'), \bar{p}_e) T_S(f'_s|f_s, T_{P_S}(s'), \bar{p}_e) T_{P_E}(\bar{p}_e) T_o(s'|s) V_{k-1}(f', f'_s, s') \quad (5)$$

$$V_k(f, f_s, s) \leftarrow \max_{o \in \mathcal{O}} Q_k(f, f_s, s, o) \quad (6)$$

B. Proofs and Conditions for Satisfaction and Optimality

The proofs are based on the more general LOF formulation of Appendix A, as results on the more general formulation also apply to the simpler formulation used in the paper.

Definition B.1. *Let the reward function of the environment be $R_{\mathcal{E}}(f_s, s, a)$, which is some combination of $R_E(s, a)$ and $R_S(f_s, \bar{p}_s) = R_S(f_s, T_{P_S}(s))$. Let $\pi' : \mathcal{F}_S \times \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ be the optimal goal-conditioned policy for reaching a state s'. In the case of a goal-conditioned policy, the reward function is $R_{\mathcal{E}}$, and the objective is to maximize the expected reward with the constraint that s' is reached in a finite amount of time. We assume that every state s' is reachable from any state s, a standard regularity assumption in MDP literature. Let $V^{\pi'}(f_s, s|s')$ be the optimal expected cumulative reward for reaching s' from s with goal-conditioned policy π'. 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 B.1. *The optimal policy for the option must be the same as the goal-conditioned policy that has subgoal s_g as its goal: $\pi^*(f_s, s) = \pi_g(f_s, s|s_g)$. In other words, $V^{\pi_g}(f_s, s|s_g) > V^{\pi'}(f_s, s|s') \;\; \forall f_s, s, s' \neq s_g$.*

This condition guarantees that the optimal option policy will always reach the subgoal s_g. It can be achieved by setting all rewards $-\infty < R_{\mathcal{E}}(f_s, s, a) < 0$ 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 B.2. *Given that the goal state of $\mathcal{W}_{liveness}$ is reachable from any other state using only subgoals and that there is an option for every subgoal and that all the options meet Condition B.1, there exists a meta-policy that can reach the FSA goal state from any non-trap state in the FSA.*

Proof. This follows from the fact that transitions in $\mathcal{W}_{liveness}$ 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, and the goal state f_g is reachable from every FSA state $f \in \mathcal{F}$ using only subgoals. □

Definition B.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 meta-policy is optimal over the available options.

Definition B.3. *Let the expected cumulative reward function of an option o started at state (f_s, s) be $R_o(f_s, s)$. Let the reward function on the SMDP be $R_{SMDP}(f, f_s, s, o) = R_F(f)R_o(f_s, s)$ with $R_F(f) \geq 0$[2]. Let $\mu' : \mathcal{F} \times \mathcal{F}_S \times \mathcal{S} \times \mathcal{O} \times \mathcal{F} \to [0, 1]$ be the hierarchically optimal goal-conditioned meta-policy for achieving liveness state f'. The objective of the meta-policy is to maximize the reward function R_{SMDP} with the constraint that it reaches f' in a finite number of time steps. Let $V^{\mu'}(f, f_s, s|f')$ be the hierarchically optimal return for reaching f' from (f, f_s, s) with goal-conditioned meta-policy mu'. 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 meta-policy for achieving the goal state.*

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

This condition guarantees that the hierarchically optimal meta-policy 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 $-\infty < R_{\mathcal{E}}(f_s, s, a) < 0 \ \forall s$, then $R_o(f_s, s) < 0 \ \forall f_s, o, s$. Then if $R_F(f) > 0$ (in our experiments, we set $R_F(f) = 1 \ \forall f$), $R_{SMDP}(f, f_s, s, o) = R_F(f)R_o(f_s, s) < 0$, 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 B.4. *From (Sutton et al., 1999): Value iteration on an SMDP converges to the hierarchically optimal policy.*

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

Definition B.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 $\mathcal{E}$, the liveness property $\mathcal{W}_{liveness}$, and the safety property $\mathcal{W}_{safety}$. Let $R_F(f) > 0 \ \forall f$, and let $R_{HMDP}(f, f_s, s, a) = R_F(f)R_{\mathcal{E}}(f_s, s, a)$, and let π^*_{HMDP} be the optimal policy for the HMDP.*

Theorem B.5. *Given Conditions B.1 and B.3, the hierarchically optimal meta-policy μ_g with optimal option policies π_g has the same expected returns as the HMDP optimal policy π^* and satisfies the task specification.*

[2]The assumption that $R_{SMDP}(f, f_s, s, o) = R_F(f)R_o(f_s, s)$ and $R_{HMDP}(f, f_s, s, a) = R_F(f)R_{\mathcal{E}}(f_s, s, a)$ 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.

Proof. By Condition B.1, every subgoal has an option associated with it whose optimal policy is to go to the subgoal. By Condition B.3, the hierarchically optimal meta-policy will reach the FSA goal state f_g. The meta-policy 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 that correspond to a satisfying sequence of subgoals and are optimal in expectation. Therefore, since $R_F(f) > 0 \ \forall f$ and $R_{SMDP}(f, f_s, s, o) = R_F(f)R_o(f_s, s)$, and since the event propositions that affect the order of subgoals necessary to satisfy the task are independent random variables, the expected cumulative reward is a positive linear combination of the expected option rewards, and since all option rewards are optimal with respect to the environment and the meta-policy is optimal over the options, our algorithm attains the optimal expected cumulative reward. □

C. Experimental Implementation

We discuss the implementation details of the experiments in this section. Because the setups of the domains are analogous, we discuss the delivery domain first in every section and then briefly relate how the same formulation applies to the reacher and pick-and-place domains as well. In this section, we use the simpler formulation of the main paper and not the more general formulation discussed in Appendix A.

C.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. 12a). The safety propositions are $\mathcal{P}_S = \{o, e\}$. o is the obstacle proposition. It is associated with many states – the black squares in Fig. 12a. 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. The event proposition is $\mathcal{P}_E = \{can\}$. can is the "cancelled" proposition, representing when one of the subgoals has been cancelled.

To simplify the FSAs and the implementation, we make an assumption 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 event 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 safety proposition e, and it also has the event proposition can

and 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 meta-policy will still converge to a hierarchically optimal policy.

The pick-and-place domain has subgoals r, g, b, y like the reacher domain, and event proposition can. Like the reacher domain, the pick-and-place domain's subgoals become true in a region around the goal state, breaking one of the necessary conditions for optimality. However, the LOF meta-policy still converges to a hierarchically optimal policy.

C.2. 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.

For the delivery domain, the 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 B.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 B.3 so that the LOF meta-policy 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 meta-policy learning. For meta-policy 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 sub-policies for FSA states, and the subgoals for FSA states are not known ahead of time, so distance to subgoal cannot be calculated.

The pick-and-place domain has reward functions analogous to the reacher domain's.

C.3. Algorithm for `LOF-QL`

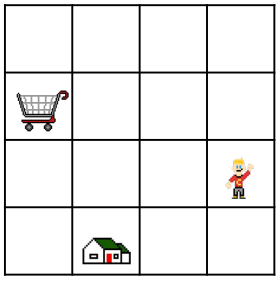
The `LOF-QL` baseline uses Q-learning to learn the meta-policy instead of value iteration. We therefore use "Logical Q-Learning" equations in place of the Logical Value Iteration equations described in Eqs. 3 and 4 in the main text. The algorithm is described in Alg. 3. A benefit of using Q-learning instead of value iteration is that the transition function T_F of the FSA $\mathcal{T}$ does not have to be explicitly known, as the algorithm samples from the transitions rather than using T_F explicitly in the formula. However, as described in the main text, this comes at the expense of reduced composability, as `LOF-QL` takes around $5x$ more iterations to converge to a new meta-policy than `LOF-VI` does. Let $Q_0(f, s, o)$ be initialized to be all 0s. The Q update formulas are given in Alg. 3 lines 13 and 14.

C.4. Comparison of LOF-VI and Q-Learning for Reward Machines

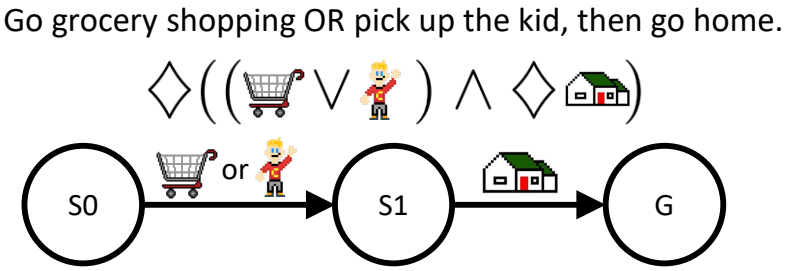
Figs. 4, 5, 6, and 7 give a visual overview of how LOF-VI and Q-Learning for Reward Machines work, and illustrate how they differ.

C.5. Tasks

We test the environments on four tasks, a "sequential" task (Fig. 8), an "IF" task (Fig. 9), an "OR" task (Fig. 10), and a "composite" task (Fig. 11). 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.



(a) Environment MDP $\mathcal{E}$.



(b) Liveness property $\mathcal{T}$. The natural language rule can be represented as an LTL formula which can be translated into an FSA.

Figure 4. LOF and RM both require an environment MDP $\mathcal{E}$ and an automaton $\mathcal{T}$ that specifies a task.

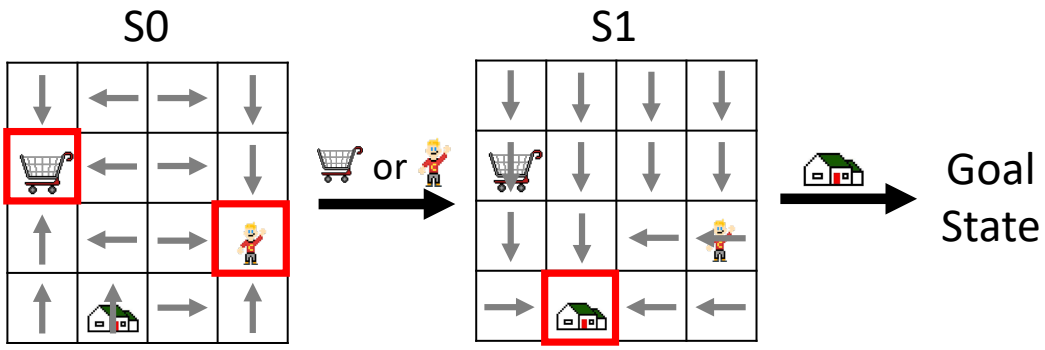


Figure 5. In RM, sub-policies are learned for each state of the automaton. In this case, in state $S0$, a sub-policy is learned that goes either to the shopping cart of the kid, whichever is closer. In state $S1$, the sub-policy goes to the house.

C.6. 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.

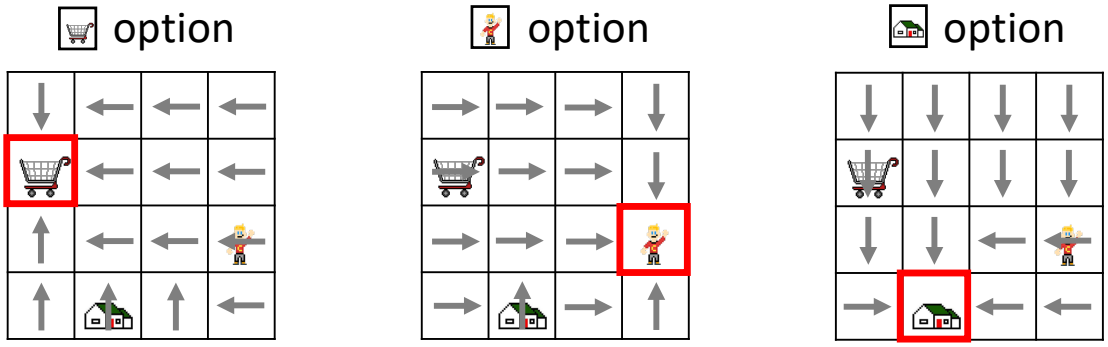
For the satisfaction experiments for the reacher domain, a single policy was trained for each task and for each baseline. The baselines were trained for 900 epochs, with 50 steps per epoch. Every 2500 training steps, each policy was tested by doing 10 rollouts and recording the returns. For the RM baseline, 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 meta-policing 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.

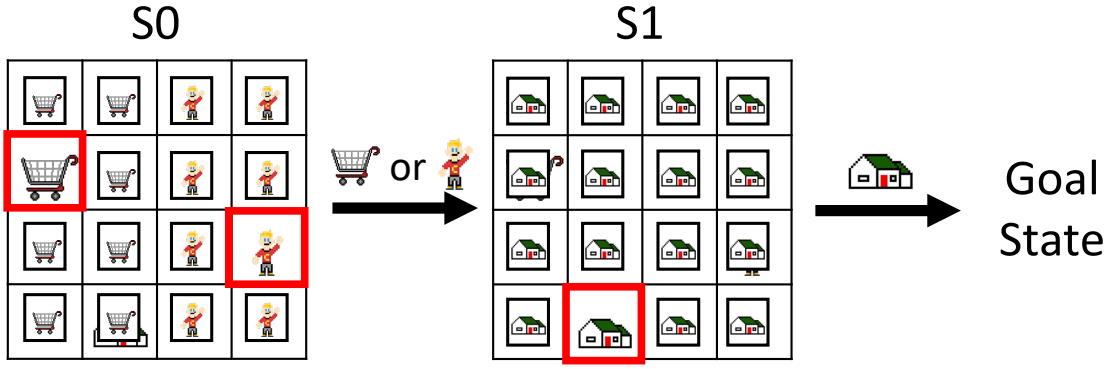
For the pick-and-place domain, 1 policy was trained for the satisfaction experiments, and experimental results were evaluated over 10 rollouts. Training was done for 7500 epochs with 1000 steps per epoch. Every 250,000 training steps, the policy was tested by doing 10 rollouts and recording the returns. For the composability experiments, returns were recorded by rolling out each baseline 2 times. The RM baseline was trained over 10000 epochs with 1000 steps per epoch.

We ran experiments on a workstation with an Intel i9 processor and an Nvidia 1080Ti GPU. The total number of training steps, size of the model, and training time of LOF-VI and RM are shown in Table 1. Information for LOF-QL, Greedy, and Flat Options are not shown because they are equivalent to LOF-VI. This is because the vast majority of the computational workload is spent training the low-level options (which are the same for LOF-VI, LOF-QL, Greedy, and Flat Options).

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



(a) Step 1 of LOF: Learn a logical option for each subgoal.



(b) Step 2 of LOF: Use Logical Value Iteration to find a meta-policy that satisfies the liveness property. In this image, the boxed subgoals indicate that the corresponding option is the optimal option to take from that low-level state. The policy ends up being the same as RM's policy – in state $S0$, the optimal meta-policy chooses the "grocery shoppping" option if the grocery cart is closer and the "pick up kid" option if the kid is closer. In the state $S1$, the optimal meta-policy is to always choose the "home" option.

Figure 6. LOF has two steps. In (a) the first step, logical options are learned for each subgoal. In (b) the second step, a meta-policy is found using Logical Value Iteration.

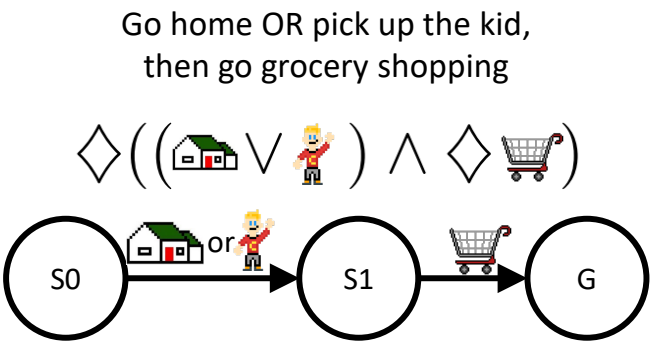
D. Further Discussion

What happens when incorrect rules are used? One benefit of representing the rules of the environment as LTL formulae/automata is that these forms of representing rules are much more interpretable than alternatives (such as neural nets). Therefore, if an agent's learned policy has bad behavior, a user of LOF can inspect the rules to see if the bad behavior is a consequence of a bad rule specification. Furthermore, one of the consequences of composability is that any modifications to the FSA will alter the resulting policy in a direct and predictable way. Therefore, for example, if an incorrect human-specified task yields undesirable behavior, with our framework it is possible to tweak the task and test the new policy without any additional low-level training (however, tweaking the safety rules would require retraining the logical options).

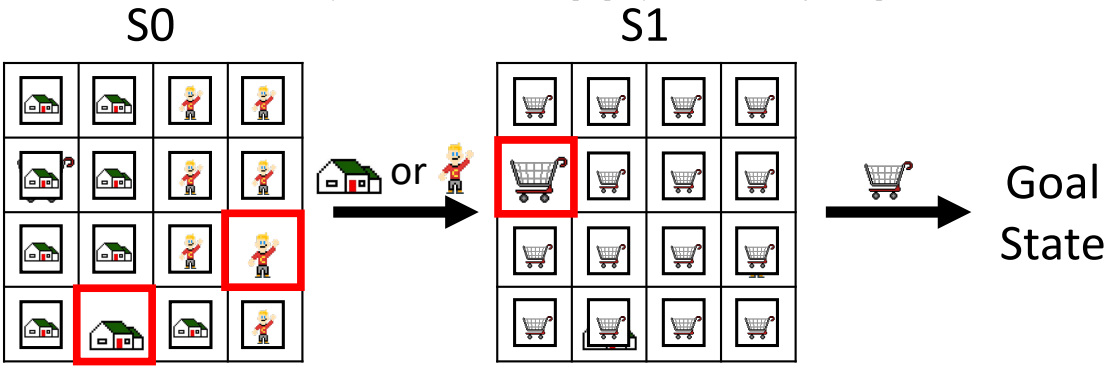
What happens if there is a rule conflict? If the specified LTL formula is invalid, the LTL-to-automaton translation tool will either throw an error or return a trivial single-state automaton that is not an accepting state. Rollouts would terminate immediately.

What happens if the agent can't satisfy a task without violating a rule? The solution to this problem depends on the user's priorities. In our formulation, we have assigned finite costs to rule violations and an infinite cost to not satisfying the task (see Appendix B). We have prioritized task satisfaction over safety satisfaction. However, it is possible to flip the priorities around by terminating training/rollouts if there is a safety violation. In our proofs, we have assumed that the agent can reach every subgoal from any state, implying either that it is always possible to avoid safety violations or that safety violations are allowed.

Why is the safety property not composable? The safety property is not composable because we allow safety propositions to be associated with more than one state in the environment (unlike subgoals). The fact that there can be multiple instances of a safety proposition in the environment means that it is impossible to guarantee that a new option policy will be optimal if retraining is done only at the level of the safety automaton and not also over the low-level states. In order to guarantee optimality, retraining would have to be done over both the high and low levels (the safety



(a) LOF can easily solve this new liveness property without training new options.



(b) Logical Value Iteration can be used to find a meta-policy on the new task without the need to retrain the logical options. A new meta-policy can be found in 10-50 iterations. The new policy finds that in state $S0$, "home" option is optimal if the agent is closer to "home", and the "kid" option is optimal if the agent is closer to "kid". In state $S1$, the "grocery shopping" option is optimal everywhere.

Figure 7. What distinguishes LOF from `RM` is that the logical options of LOF can be easily composed to solve new tasks. In this example, the new task is to go home or pick up the kid, then go grocery shopping. Logical Value Iteration can find a new meta-policy in 10-50 iterations without needing to relearn the options.

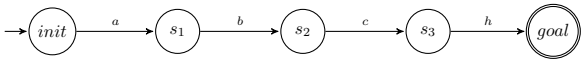


Figure 8. FSA for the sequential task. The LTL formula is $\Diamond(a \wedge \Diamond(b \wedge \Diamond(c \wedge \Diamond h))) \wedge \Box!o$. The natural language interpretation is "Deliver package a, then b, then c, and then return home h. And always avoid obstacles o".

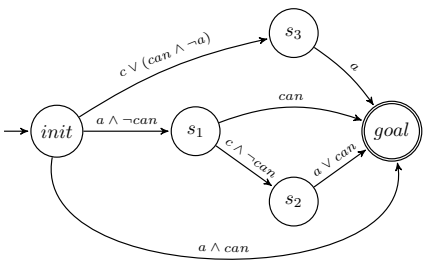


Figure 9. FSA for the IF task. The LTL formula is $(\Diamond(c \wedge \Diamond a) \wedge \Box!can) \vee (\Diamond a \wedge \Diamond can) \wedge \Box!o$. The natural language interpretation is "Deliver package c, and then a, unless a gets cancelled. And always avoid obstacles o".

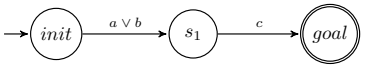


Figure 10. FSA for the OR task. The LTL formula is $\Diamond((a \vee b) \wedge \Diamond c) \wedge \Box!o$. The natural language interpretation is "Deliver package a or b, then c, and always avoid obstacles o".

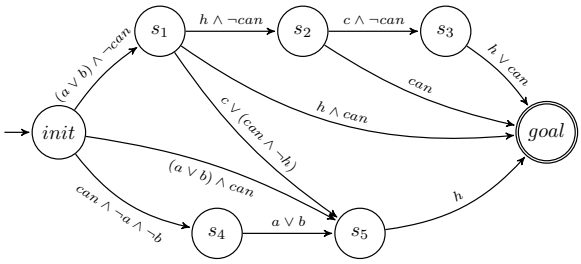
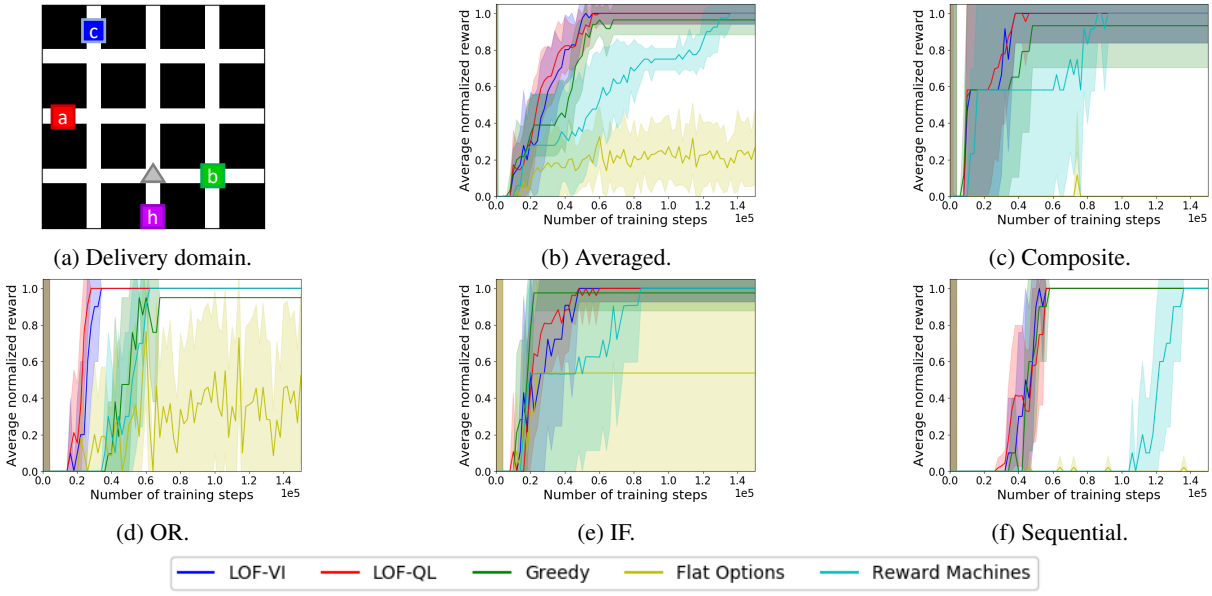


Figure 11. FSA for the composite task. The LTL formula is $(\Diamond((a \vee b) \wedge \Diamond(c \wedge \Diamond h)) \wedge \Box!can) \vee (\Diamond((a \vee b) \wedge \Diamond h) \wedge \Diamond can) \wedge \Box!o$. The natural language interpretation is "Deliver package a or b, and then c, unless c gets cancelled, and then return to home h. And always avoid obstacles".

Domain	Algorithm	Epochs	Steps per epoch	Total training steps	Size of model	Training time (sec)	Training time (hours)
Delivery	LOF-VI	1600	100	160,000	~36K	29.2	0.0081
Delivery	RM	1600	100	160,000	~92K	44.1	0.012
Reacher	LOF-VI	900	50	45,000	154K	2200	0.62
Reacher	RM	1000	800	800,000	155K	11300	3.14
Pick and place	LOF-VI	7500	1000	7,500,000	279K	40900	11.35
Pick and place	RM	10000	1000	10,000,000	297K	55500	15.41

Table 1. Information on experimental tests for the LOF-VI and RM algorithms. Info for LOF-QL, Greedy, and Flat Options is not shown because it is equivalent to that for LOF-VI. This is because the training of the low-level options takes up the vast majority of training time and accounts for all of the size of the model. Model sizes for the delivery domain are approximate as they vary with the number of FSA states of the liveness property.



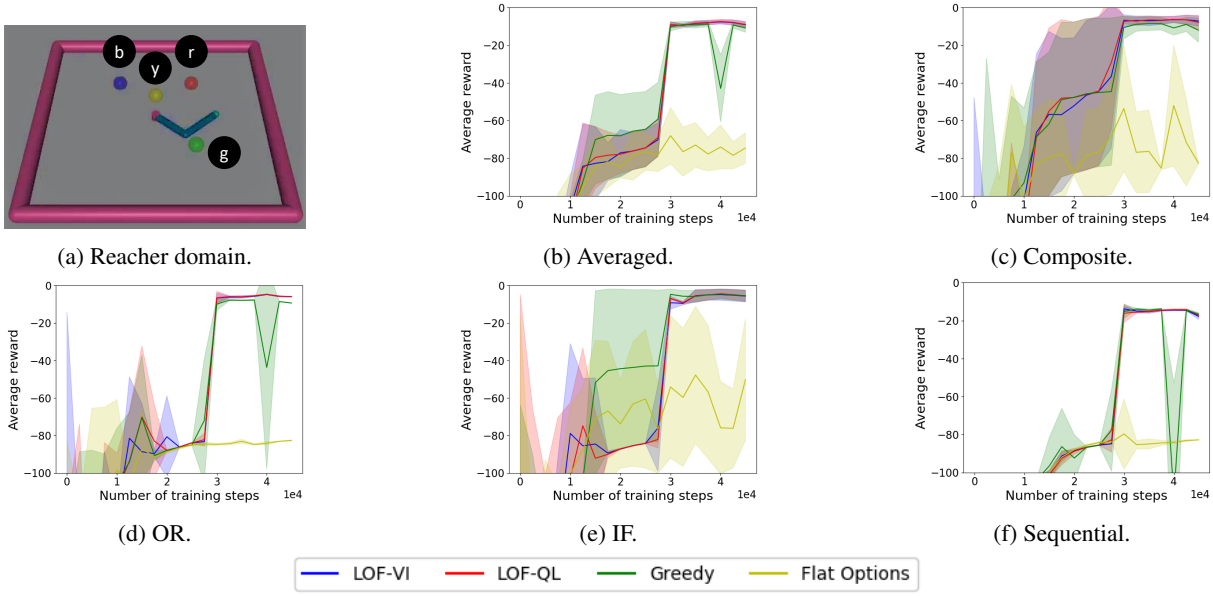
(a) Delivery domain. (b) Averaged. (c) Composite.

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

Figure 12. All satisfaction experiments on the delivery domain. Notice how for the composite and OR tasks (Figs. 12c and 12d), 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. 12f), 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.

automaton and the environment). Our definition of composability involves only replanning over the high level of the FSA. Therefore, safety properties are not composable. Furthermore, rewards/costs of the safety property can be associated with propositions and not just with states (as with the liveness property). This is because a safety violation via one safety proposition (e.g., a car going onto the wrong side of the road) may incur a different penalty than a violation via a different proposition (a car going off the road). The propositions are associated with low-level states of the environment. Therefore any retraining would have to involve retraining at both the high and low levels, once again violating our definition of composability.

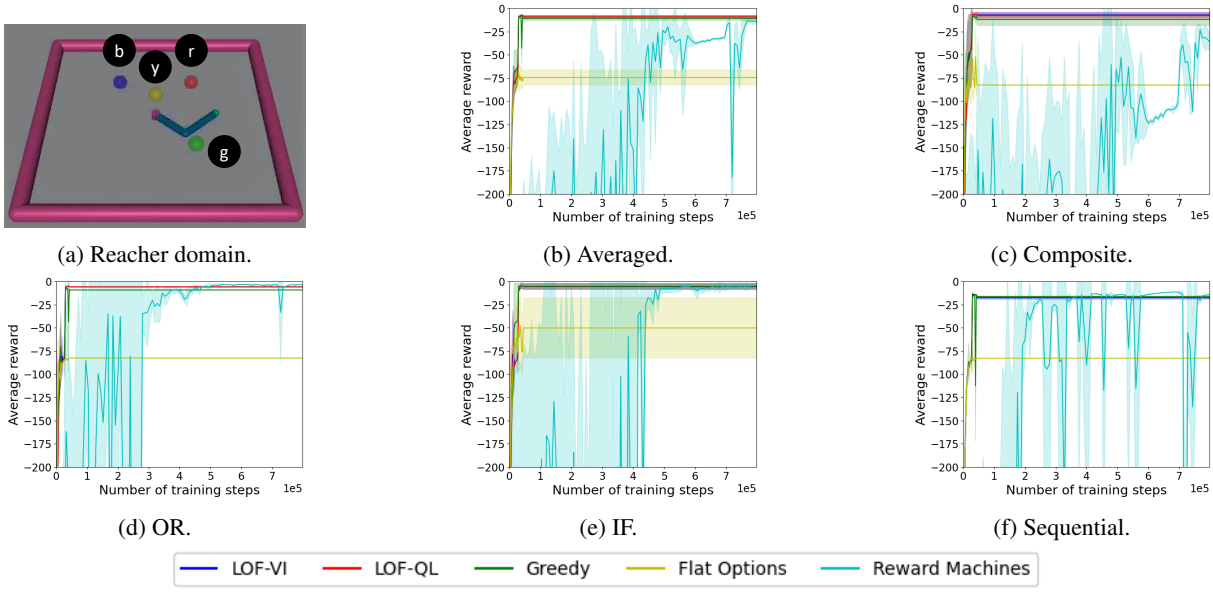
Simplifying the option transition model: In our experiments, we simplify the transition model by setting $\gamma = 1$, an assumption that does not affect convergence to optimality. In the case where $\gamma = 1$, Eq. 2 reduces to $T_o(s'|s) = \sum_k p(s', k)$. Assuming that the option terminates only at state s_g, then Eq.2 further reduces to $T_o(s_g|s) = 1$ and $T_o(s'|s) = 0$ for all other $s' \neq s_g$. Therefore no learning is required for the transition model. For cases where the assumption that $\gamma = 1$ does not apply, (Abel & Winder, 2019) contains an interesting discussion.



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

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

Figure 13. Satisfaction experiments for the reacher domain, without RM results. The results are equivalent to the results on the delivery domain.

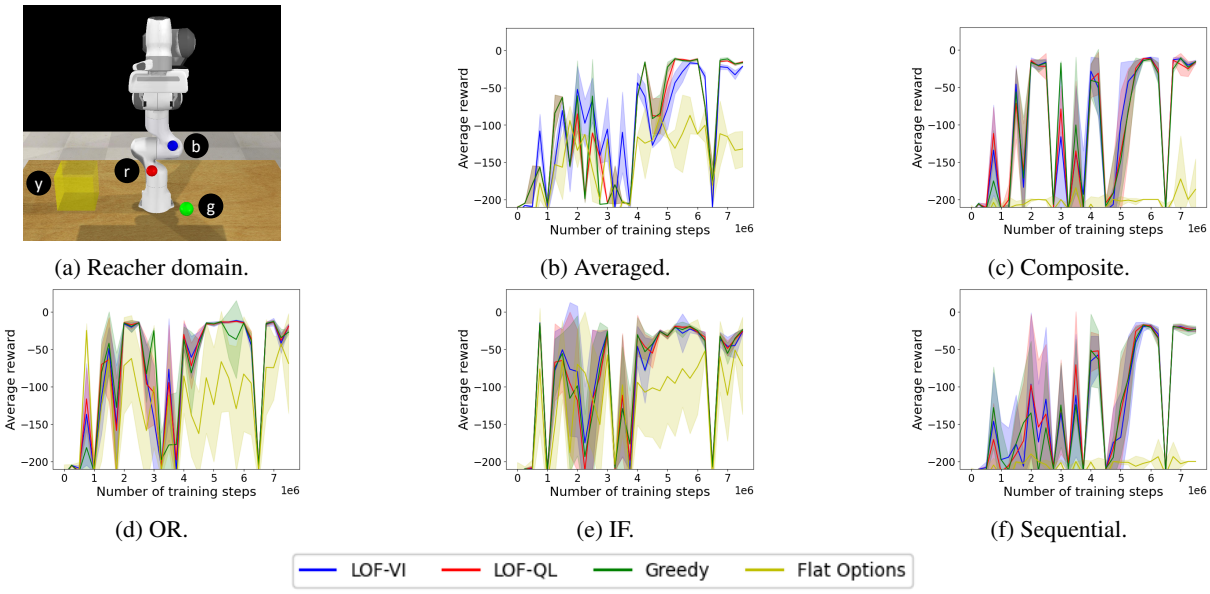


(a) Reacher domain. (b) Averaged. (c) Composite.

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

Figure 14. Satisfaction experiments for the reacher 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.

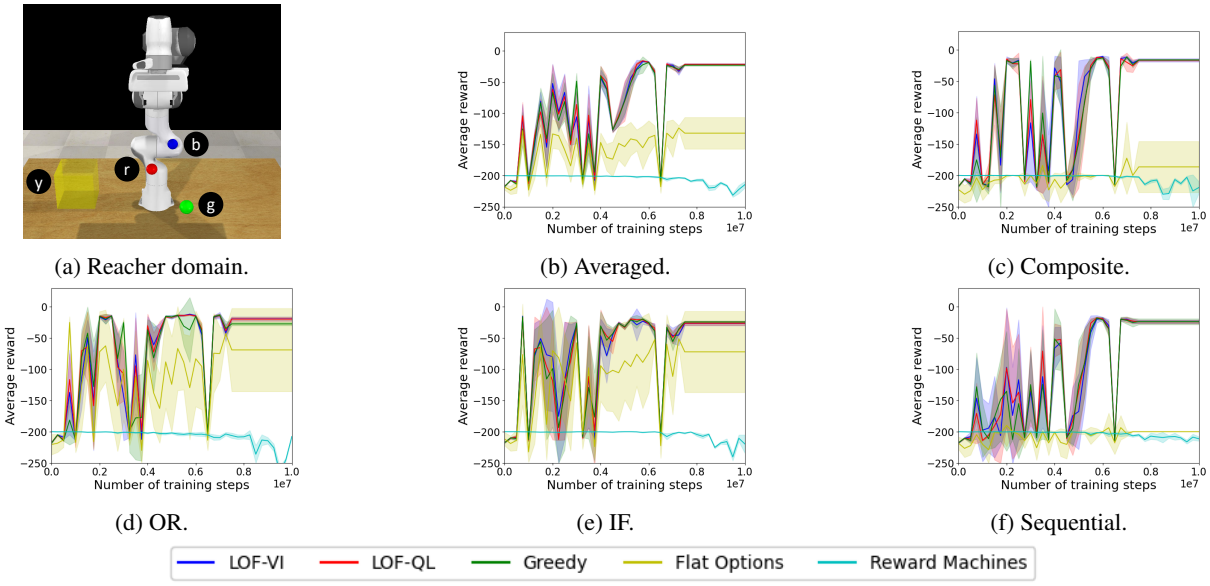
Learning the option reward model: The option reward model $R_o(s)$ is the expected reward of carrying out option o to termination from state s. It is equivalent to a value function. Therefore, it is convenient if the policy-learning algorithm used to learn the options learns a value function as well as a policy (e.g., Q-learning and PPO). However, as long as the expected return can be computed between pairs of states, it is not necessary to learn a complete value



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

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

Figure 15. Satisfaction experiments for the pick-and-place domain, without RM results. The results are equivalent to the results on the delivery and reacher domains.



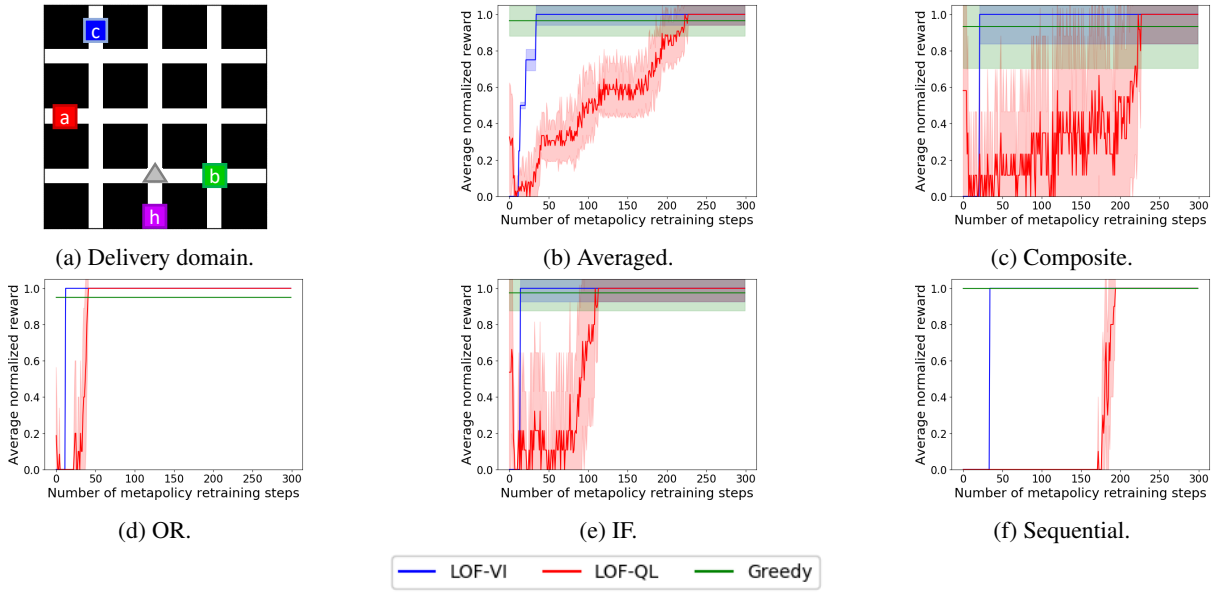
(a) Reacher domain. (b) Averaged. (c) Composite.

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

Figure 16. Satisfaction experiments for the pick-and-place domain, including RM results. For the pick-and-place domain, RM did not converge to a solution within the training time allotted for it (10 million training steps).

function. This is because during Logical Value Iteration, the reward model is only queried at discrete points in the state space (typically corresponding to the initial state and the subgoals). So as long as expected returns between the initial state and subgoals can be computed, Logical Value Iteration will work.

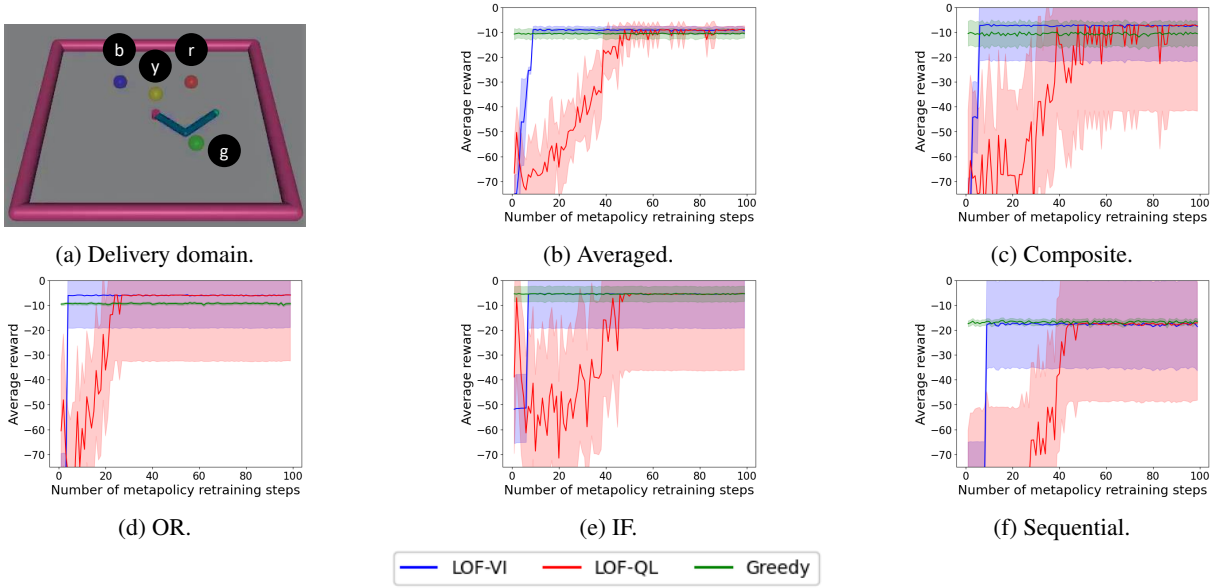
Why is LOF-VI so much more efficient than the RM baseline? In short, LOF-VI is more efficient than RM because LOF-VI has a dense reward function during training and RM has a sparse reward function. During training, LOF-VI trains the options independently and rewards the agent for reaching the subgoals associated with the options. This is in effect a dense reward function. The generic reward function for RM only rewards the agent for reaching the goal state. There are no other high-level rewards to guide the agent through the task. This is a very sparse reward that results in less efficient training. RM's reward function



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

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

Figure 17. All composability experiments for the delivery domain.

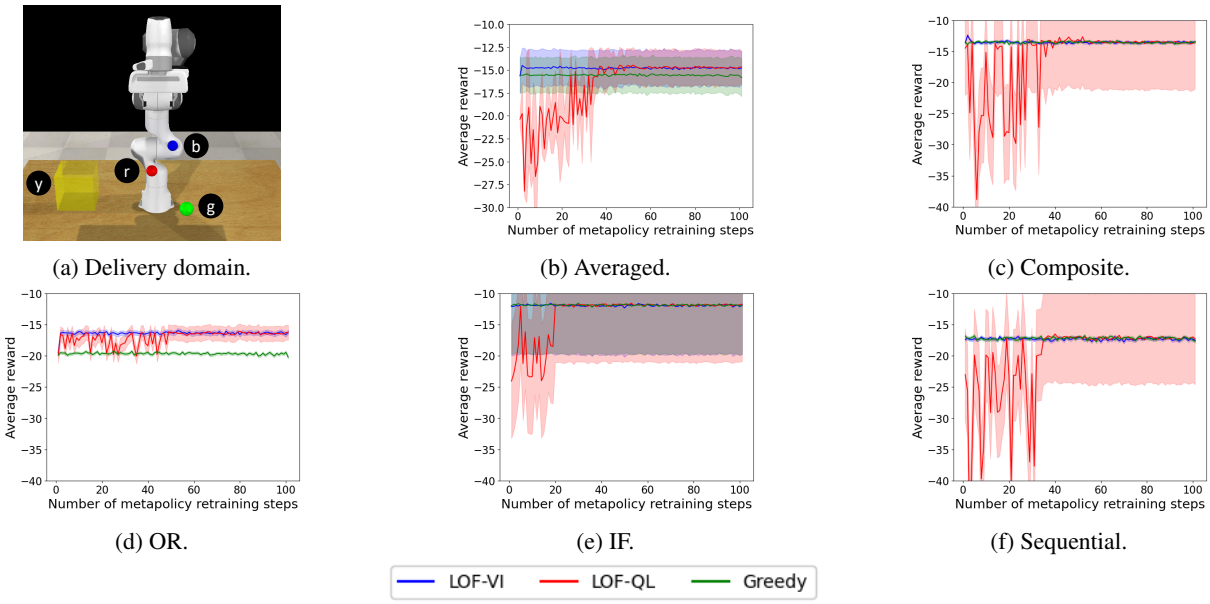


(a) Delivery domain. (b) Averaged. (c) Composite.

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

Figure 18. All composability experiments for the reacher domain.

could easily be made dense by rewarding every transition of the automaton. In this case, RM would probably train as efficiently as LOF-VI. However, imagine an FSA with two paths to the goal state. One path has only 1 transition but has much lower low-level cost, and one path has 20 transitions and a much higher low-level cost. RM might learn to prefer the reward-heavy 20-transition path rather than the reward-light 1-transition path, even if the 1-transition path results in a lower low-level cost. In theory it might be possible to design an RM reward function that adjusts the automaton transition reward depending on the length of the path that the state is in, but this would not be a trivial task when accounting for branching and merging paths. We therefore decided that it would be a fairer comparison to use a trivial RM reward function, just as we use a trivial reward function for the LOF baselines. However, we were careful to not list increased efficiency in our list of contributions; although increased efficiency was an observed side effect of LOF, LOF is not inherently more efficient than other algorithms besides the fact that it automatically imposes a



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

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

Figure 19. All composability experiments for the pick-and-place domain.

dense reward on reaching subgoals.

Algorithm 3 LOF with ϵ-greedy Q-learning

Given:
Propositions $\mathcal{P}$ partitioned into subgoals $\mathcal{P}_G$, safety propositions $\mathcal{P}_S$, and event propositions $\mathcal{P}_E$
Environment MDP $\mathcal{E} = (\mathcal{S}, \mathcal{A}, T_E, R_\mathcal{E}, \gamma)$
Logical options $\mathcal{O}$ with reward models $R_o(s)$ and transition models $T_o(s'|s)$
Liveness property $\mathcal{T} = (\mathcal{F}, \mathcal{P}_G \cup \mathcal{P}_E, T_F, R_F, f_0, f_g)$
(T_F does not have to be explicitly known if it can be sampled from a simulator)
Learning rate α, exploration probability ϵ
Number of training episodes n, episode length m
To learn:
Meta-policy $\mu(f, s, o)$ along with $Q(f, s, o)$ and $V(f, s)$
Find a meta-policy μ over the options:
Initialize $Q : \mathcal{F} \times \mathcal{S} \times \mathcal{O} \to \mathbb{R}$ and $V : \mathcal{F} \times \mathcal{S} \to \mathbb{R}$ to 0
for $k \in [1, \ldots, n]$: **do**
  Initialize FSA state $f \leftarrow 0$, s a random initial state from $\mathcal{E}$
  Draw $\bar{p}_e \sim T_{P_E}()$
  for $j \in [1, \ldots, m]$: **do**
    With probability ϵ let o be a random option; otherwise, $o \leftarrow \arg\max_{o' \in \mathcal{O}} Q(f, s, o')$
    $s' \sim T_o(s)$
    $f' \sim T_F(T_{P_G}(s'), \bar{p}_e, f)$
    $Q_k(f, s, o) \leftarrow Q_{k-1}(f, s, o) + \alpha\big(R_F(f)R_o(s) + \gamma V(f', s') - Q_{k-1}(f, s, o)\big)$
    $V_k(f, s) \leftarrow \max_{o' \in \mathcal{O}} Q_k(f, s, o')$
    $f \leftarrow f'$
  end for
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 $Q(f, s, o), V(f, s)$