Run-Time Task Composition with Safety Semantics

Kevin Leahy [*1] **Makai Mann** [*2] **Zachary Serlin** [2]

Abstract

Compositionality is a critical aspect of scalable system design. Here, we focus on Boolean composition of learned tasks as opposed to functional or sequential composition. Existing Boolean composition for Reinforcement Learning focuses on reaching a satisfying absorbing state in environments with discrete action spaces, but does not support composable safety (i.e., avoidance) constraints. We provide three contributions: i) introduce two distinct notions of compositional safety semantics; ii) show how to enforce either safety semantics, prove correctness, and analyze the trade-offs between the two safety notions; and iii) extend Boolean composition from discrete action spaces to continuous action spaces. We demonstrate these techniques using modified versions of value iteration in a grid world, Deep Q-Network (DQN) in a grid world with image observations, and Twin Delayed DDPG (TD3) in a continuous-observation and continuous-action Bullet physics environment.

1. Introduction

Recent advances have established reinforcement learning (RL) as a powerful tool for human-level performance in games (Schrittwieser et al., 2020), robotics (Ibarz et al., 2021), and other fields (Arulkumaran et al., 2017). However, there are still many technical hurdles to deploying these algorithms in real-world scenarios. For many of these approaches, millions of samples are required before the desired behavior is achieved. Additionally, there are concerns about side effects, reward hacking, and transparency (Amodei et al., 2016). Transfer learning (Taylor & Stone, 2009), and in particular task composition (Nangue Tasse et al., 2020), has emerged as a promising method for learning simple task primitives and composing them in a zero-shot manner to perform more complex behaviors. Composing tasks in this way ensures that if desired behavior is achieved at the level of task primitives, then desired behavior will be achieved when tasks are composed.

Prior work in composition has focused primarily on reachability problems, in the form of stochastic shortest paths (Nangue Tasse et al., 2020; 2022a;b). However, another important property is safety. That is, avoiding undesirable states on the way to reaching a final state. Here, we present a method for composing simple policies to satisfy complex behaviors with no additional training that also include zero-shot safety considerations. To illustrate this distinction, we include the following motivating example.

Example 1. Consider an autonomous waste management vehicle, which picks up from M residential or commercial zones, and drops off at one of N processing plants with a combination of capabilities drawn from trash, recycling, and hazardous waste processing. If carrying hazardous waste, the truck must avoid residential zones.

One approach is to train policies for each of the M pick up areas, plus two policies for each of the N plants–one that avoids residential zones and one that does not, resulting in $M + 2N$ policies. Our method instead trains a single policy for each type of the five zone types: residential, commercial, trash, recycling, and hazardous waste. We also add an additional policy for avoiding residential zones, resulting in six total policies. These policies can be composed at runtime. For example, a truck carrying both trash and recycle can compose the trash and recycling policies to arrive at a processing plant that handles both. Furthermore, the residential avoidance policy can be composed via conjunction when the truck is carrying hazardous waste. Notice that residential areas are sometimes goals and sometimes obstacles.

Safety-aware learning has become of great interest in recent years for learning-based systems in safety-critical applications (e.g., autonomous driving). We combine components from both the safety-aware learning and task composition communities to create safety-aware task composition.

Contributions: Our main contribution is a method for generating and composing multiple safety-aware policies using Q-learning approaches that can be combined at deployment using the presented safety-aware Boolean task algebra for-

[*]Equal contribution [1]Department of Robotics Engineering, Worcester Polytechnic Institute, Lexington, MA, USA [2]MIT Lincoln Laboratory, Lexington, MA, USA. Correspondence to: Kevin Leahy <kleahy@wpi.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

mulation. Specifically, (1) we present two new semantics to create safety-aware Boolean task algebra that encode safety constraints as (a) minimally entering states that are not explicitly goal states (by count) and (b) minimally entering states that are prioritized as bad states; (2) we show how to enforce either safety semantics, prove correctness under some assumptions, and analyze the trade-offs between the two safety notions; and (3) we extend Boolean composition from discrete action spaces to continuous action spaces. We demonstrate these techniques using modified versions of value iteration in a grid world, Deep Q-Network (DQN) in a grid world with image observations, and Twin Delayed DDPG (TD3) in a continuous-observation and continuous-action Bullet physics simulation environment.

Related work: This paper is most closely related to Nangue Tasse et al. (2020) and its extensions (Nangue Tasse et al., 2022a;b). In Nangue Tasse et al. (2020), the authors present a Boolean task algebra for composing simple discrete Q-tables. While that work considers Boolean composition, including negation, it only focuses on reachability tasks. Negating a task in that context only means that an agent will not terminate (by selecting a dedicated terminal action) in a goal region associated with that task. However, a more standard semantics for negation is *avoidance*. Let us call theirs "reachability-only" (RO) semantics as opposed to our "safety-aware" semantics.

Consider the truck in Example 1. It should avoid residential neighborhoods entirely when carrying hazardous waste. Negation with RO semantics only prevents the truck from *ending* its route in a residential neighborhood. Standard avoidance cannot be captured in this framework, unless passing through residential neighborhoods is *always* forbidden. Specifically, regions cannot switch between goals and avoid zones with RO semantics. In this simple example, one could add an extra Boolean state variable indicating whether the truck is carrying hazardous waste and always disallow passing through residential neighborhoods in that case. However, due to the curse of dimensionality, this does not scale well if there are more complex conditions.

Safety constraints have long existed in the formal methods and controls communities (Pnueli, 1977; Baier & Katoen, 2008; Ames et al., 2019), but have recently been applied to RL as well (Berducci et al., 2021; Dawson et al., 2023). The prevailing approaches in safe RL literature constrain the underlying Markov decision process to rule out unsafe behaviors (Alshiekh et al., 2018; Achiam et al., 2017; Tessler et al., 2019; Yang et al., 2020). These approaches permanently preclude agents from visiting regions deemed unsafe. Safe RL focuses on how to *train* safe policies, whereas our focus is on how to *compose* safe policies. Critically, our approach does not require unsafe regions to be declared up front. Rather, we train composable policies such that regions can dynamically be marked as desirable or undesirable at runtime. Future work may leverage the orthogonal techniques developed in safe RL literature to improve the quality and training time of our composable policies at the task primitive level.

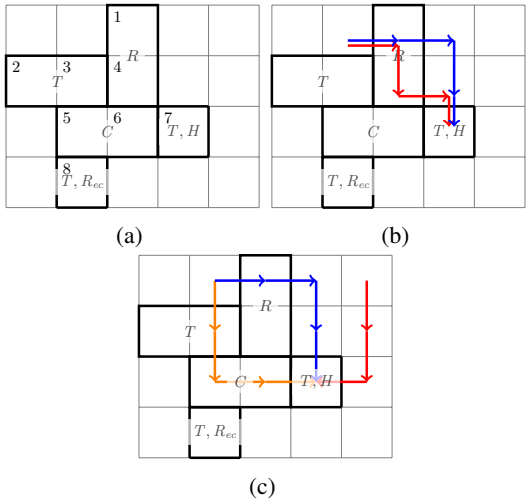


Figure 1: Environment inspired by Example 1 with regions T=trash, R_{ec}=recycle, H=hazardous waste, C=commercial, and R=residential. 1a: $\mathcal{P} = \{T, R_{ec}, H, C, R\}$. $\mathcal{G}$ consists of states 1-8. 1b: Two sample paths through the environment. The associated sequence of labels for both paths is $\{\emptyset, R, \emptyset, \emptyset, \{T, H\}\}$, despite the differing paths. 1c: Different types of paths that all satisfy $\phi = T \wedge H$. A *pure* path in red, a *minimum-violation* (MV) path in blue, and a *prioritized safety* (PS) path in orange for $\phi = T \wedge H \wedge \neg R$. Note that the MV and PS paths start in the same region, but the PS path is much longer to avoid producing the label R.

Existing prior work synthesizes rewards that enable both safety and reachability according to hierarchical structures (Berducci et al., 2021) or specification languages (Jothimurugan et al., 2019; 2021; Žikelić et al., 2023). However, these approaches are not designed to perform compositionally at run-time. That is, to combine two tasks, a new policy needs to be trained. Further, there has been an effort to address more general forms of policy composition (Adamczyk et al., 2023). Such work provides bounded optimality on the arbitrary composition of tasks, but does not provide guarantees on exact zero-shot composition.

2. Problem formulation

2.1. Markov decision processes

Let $\mathcal{P}$ be a set of atomic propositions to be used for labeling regions of the state space. We model an agent's environment (and its motion in the environment) as a *deterministic* labeled Markov decision process (MDP). From now on, we

use MDP to refer to labeled MDPs unless otherwise noted. An MDP is written as a tuple $\langle \mathcal{S}, \mathcal{A}, \tau, R, L \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\tau : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition function, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, and $L : \mathcal{S} \to 2^{\mathcal{P}}$ is a labeling function mapping each state to a set of atomic propositions.

An *execution* of an MDP is a finite sequence of tuples $x = \langle s_0, \emptyset \rangle, \langle s_1, l_1 \rangle, \ldots$, where $s_i \in \mathcal{S}$ are states and $l_i \in 2^{\mathcal{P}}$ are labels. We use l_i to denote label variables, and A, B, C to denote arbitrary labels. We consider MDPs that only emit symbols when the label given by L changes. The first label in an execution is always the empty set. This behavior is illustrated in Fig. 1b. The red path and the blue path produce the same sequence of labels. Although the red path passes through two states labeled R, the second state produces $\emptyset$, since it shares the same label as the previous state.

We further introduce the *projection* of an execution $\restriction_L : \mathcal{S} \times 2^{\mathcal{P}} \to 2^{\mathcal{P}}$, that projects an execution onto the set of associated labels. That is, for an execution $x = \langle s_0, l_0 \rangle, \langle s_1, l_1 \rangle, \ldots$, we write $\restriction_L(x) = l_0, l_1, \ldots$. We denote the sequence of non-empty symbols from the projection as $\restriction_L^+$. For example, if $\restriction_L(x) = \emptyset, \emptyset, A, \emptyset, \{A, B\}, \emptyset, B$, then $\restriction_L^+(x) = A, \{A, B\}, B$. Let $|\restriction_L|$ and $|\restriction_L^+|$ denote the length of each of these projections.

The labeling function of an MDP induces a set $\mathcal{G} \subseteq \mathcal{S}$, consisting of all labeled states of $\mathcal{S}$. We refer to these as *goals*. Any state not belonging to any $\mathcal{G}$ is unlabeled and maps to the empty set.

We can inductively define Boolean formulas over $\mathcal{P}$ as

$$\phi := p \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \,, \tag{1}$$

where $p \in \mathcal{P}$ is a proposition; ϕ is a formula; and $\neg$, $\wedge$, and $\vee$ are the standard Boolean operations of negation, conjunction, and disjunction, respectively.

For a given Boolean formula ϕ over $\mathcal{P}$, we say that an execution x of an MDP satisfies the formula (written $x \models \phi$) if the last element of $\restriction_L^+(x)$ satisfies the formula (in the Boolean sense). E.g., if the associated task is A, as long as the last element of $\restriction_L^+(x)$ contains the symbol A, then the task is satisfied. We assume there is a reserved action for terminating the trace. Note that in Nangue Tasse et al. (2020) semantics, a symbol is only produced at the *end* of the trace. Thus in their semantics, every trace x has $|\restriction_L^+(x)| = 1$.

For a Boolean formula ϕ, we denote the set of all goal states satisfying this formula as $\mathcal{G}_\phi := \{g \in \mathcal{G} \mid L(g) \models \phi\}$. We refer to a contiguous set of goal states with the same label l as a region. Note that by our labeled MDP semantics, the agent only receives symbols l upon entering a region. $\mathcal{G}$ can be partitioned into regions. Fig. 1a illustrates a labeled environment and shows the differences between $\mathcal{P}$ and $\mathcal{G}$.

2.2. Safety properties

In this work, we want proper policies–policies that are guaranteed to reach an satisfying state given some specification ϕ–that have additional properties. Namely, they should avoid passing through unsafe (or undesired) states. We consider three sub-classes of proper paths in an MDP:

1. *Pure* paths – proper paths that do not produce any symbols except the satisfying symbols at the end of the execution;
2. *Minimum-violation* (MV) paths – proper paths that produce the minimum number of other symbols; and
3. *Prioritized safety* (PS) paths – proper paths that completely avoid certain regions, and produce an MV path over the remaining regions.

Motivation: These safe path definitions are inspired by temporal logic (TL) planning. To accomplish a task specified in Linear Temporal Logic (Pnueli, 1977), an agent must satisfy Boolean formulae to take transitions in a (Büchi) automaton. Satisfying any *different* formula first could lead to an unintended transition that violates the specification. It is often important to satisfy *only* the desired formula. See Belta et al. (2017) for details on planning under TL constraints. RO semantics will not attempt to avoid unintended transitions.

Intuitively, if a specification is not available at training time, a pure path is the most desirable. Such paths terminate at the desired states and produce no extraneous (non-satisfying) symbols. MV paths prioritize reachability goals, while minimizing extraneous symbols as much as possible. PS paths, on the other hand, place higher weight on avoiding certain symbols. Consider $\phi := T \wedge H \wedge \neg R$. If there is no pure path to a $\{T, H\}$ state, MV would take the path that produces the fewest non-satisfying symbols, whereas PS would take a longer path to avoid producing R symbols as depicted in Fig. 1c. We define two separate semantics for safety with solutions to satisfy each type of safety property in the sequel, and discuss trade-offs for each type of safety in Sec. 3.2.

Formally, we define these paths with respect to a Boolean formula ϕ over $\mathcal{P}$ as follows.

Definition 1 (Pure path). An execution x produces a *pure* path if $|\restriction_L^+(x)| = 1$ and $x \models \phi$.

Definition 2 (Minimum-violation path). An execution x produces a *minimum-violation* (MV) path if $|\restriction_L^+(x)| > 1$ and $x \models \phi$, and there is no execution x' such that $|\restriction_L^+(x')| < |\restriction_L^+(x)|$.

Definition 3 (Safety path). An execution x produces a *safety* path if, for some bad formula ϕ_B, $x \models \phi$ and no finite prefix of x satisfies ϕ_B.

Definition 4 (Prioritized safety path). An execution x produces a *prioritized safety path* (PS) if it is a safety path, and there is no execution x' that is also a safety path, such that $|\restriction_L^+(x')| < |\restriction_L^+(x)|$.

3. Penalty-enforced safety

To enforce the semantics described in Defs. 1–4, we incorporate penalties on the production of labels that do not satisfy the current task. These include penalties for passing through regions that should be avoided, as well as for terminating in regions that are undesired. Intuitively, we will craft the reward functions such that there is a hierarchy of bad behaviors to be avoided. Less bad behaviors will be taken to avoid worse behaviors whenever possible. Worse behaviors will have a larger penalty, and each increase in the penalty must be sufficiently large to preserve the ordering. To enforce our penalty hierarchy, we use a *penalty multiplier* η. We can create a hierarchy of increasingly bad penalties by multiplying by η (or an upper bound of η). The hierarchy can extend indefinitely using real numbers, or until there is risk of underflow for machine representations. The multiplier must be sufficiently large to penalize behaviors such as early termination or passing through unsafe regions.

First, we derive the value of η, which will be used to prove our main results later in the paper. This value of η is necessary for theoretical proofs, but in practice a smaller value can often be used to achieve the same results.

Let Φ be the set of all achievable specifications in a given environment. For $\phi_1, \phi_2 \in \Phi$, and proposition $q \in \mathcal{P}$, let $AvoidPathLen(G_{\phi_1}, G_{\phi_2}, q)$ return the length of the longest of all shortest paths between any state $s_1 \in G_{\phi_1}$ and state $s_2 \in G_{\phi_2}$ that passes through the minimum number of q-regions. Similarly, $AvoidPathLen(G_{\phi_1}, G_{\phi_2}, \neg q)$ does the same but avoiding $\neg q$-regions. In English, this is the maximum number of steps *required* to get from any state in a ϕ_1-region to any state in a ϕ_2-region while passing through a minimum number of obstacles defined by q or $\neg q$.

Definition 5. We define a *penalty multiplier* $\eta \in \mathbb{N}$ by:

$$\begin{aligned}\eta := argmin_N [\forall \phi_1, \phi_2 \in \Phi, q \in \mathcal{P} \,.\\ N \geq AvoidPathLen(G_{\phi_1}, G_{\phi_2}, q) \wedge \\ N \geq AvoidPathLen(G_{\phi_1}, G_{\phi_2}, \neg q)]\end{aligned}$$

Intuitively, η should be set so that it is larger than the longest of all shortest paths. That way, a longer path with a lower penalty for every step of the path is preferred to a shorter path that incurs even one worse penalty.

Remark 3.1. In an RL setting we do not know η exactly, because we do not know the length of all paths. However, in practice we can either estimate η from interactions with the environment, set it to a very large number, or set it heuristically. Setting η to a conservative (i.e., very large negative) value results in longer time to convergence. Fortunately, in a deterministic MDP, η has a very intuitive meaning: it is the number of steps an agent may detour in order to avoid undesirable regions.

Table 1 depicts the penalty hierarchy achieved with successive multiplication by η. Notice that the worst pass-through penalty and the bad termination penalty are identical, otherwise, each penalty is worse by a factor of η. This means that it is better to incur a lighter penalty η times, than to incur the next worse penalty once.

Table 1: Penalty hierarchy

Symbol	Penalty type	Value
R_{step}	step	R_{step} (ex. -0.1)
$R_{badstep}$	bad pass through	ηR_{step}
$R_{worststep}$	worst pass through	$\eta^2 R_{step}$
$R_{badterm}$	bad termination	$\eta^2 R_{step}$
$R_{worstterm}$	worst termination	$\eta^3 R_{step}$

Given these penalties, we denote the reward function for a given task as $R_p(s_i, a, g, s_{i+1})$, which takes values:

$$\begin{cases} R_{worstterm}, & \text{if } s_{i+1} \neq g \wedge \texttt{done} \\ R_{badstep}, & \text{if } s_{i+1} \neq g \wedge \neg\texttt{done} \\ R_{badterm}, & \text{if } s_{i+1} = g \wedge p \notin l_i \wedge \texttt{done} \\ R_{goal}, & \text{if } s_{i+1} = g \wedge p \in l_i \wedge \texttt{done} \\ R_{step}, & \text{otherwise} \,, \end{cases} \tag{2}$$

where done denotes the end of an episode, $g \in \mathcal{G}$ and $p \in \mathcal{P}$ are defined in Sec. 2.1, and R_{goal} is positive.

3.1. Penalties for prioritized safety

PS semantics have a stricter notion of avoidance, and therefore they require a slightly different reward structure than the one presented in (2). For PS semantics, we instead train negated policies directly with additional penalties for $\mathcal{G}_{\neg p}$. Since we do not use a negation operator in this context, we cannot negate arbitrary Boolean formulas over tasks; however, any Boolean formula can be reduced to negation normal form (NNF). In NNF, negation only appears before literals (in this case tasks), thus we can represent any Boolean formula as conjunctions and disjunctions over positive and negated tasks. The reward function for a negated task is written as $R_{\neg p}(s_i, a, g, s_{i+1})$ and takes values:

$$\begin{cases} R_{worstterm}, & \text{if } s_{i+1} \neq g \wedge \texttt{done} \\ R_{worststep}, & \text{if } p \in l_i \wedge \neg\texttt{done} \\ R_{badstep}, & \text{if } s_{i+1} \neq g \wedge p \notin l_i \wedge \neg\texttt{done} \\ R_{badterm}, & \text{if } s_{i+1} = g \wedge p \in l_i \wedge \texttt{done} \\ R_{goal}, & \text{if } s_{i+1} = g \wedge p \notin l_i \wedge \texttt{done} \\ R_{step}, & \text{otherwise} \end{cases} \tag{3}$$

There are two main differences from the reward structure for a positive task. First, the conditions for obtaining R_{goal} and $R_{badterm}$ are switched. This is because the goal is to

terminate somewhere that does not satisfy p. Additionally, we add a penalty for passing through any state labeled with p. This penalty encourages paths that pass through regions labeled with other symbols (if necessary) rather than passing through regions containing p.

3.2. Theoretical analysis and comparison of policies

With the reward structures described above, we wish to prove the following for both MV and PS semantics:

1. If a pure path exists, the optimal policy will select it;
2. If a pure path does not exist, the optimal policy will follow a MV (resp. PS) path; and

We now introduce theorems formalizing these properties.

Theorem 1. The reward structure in (2) produces MV paths.

Proof. We introduce the following notation for the purposes of this proof and the proofs of subsequent theorems.

Variables that capture path length:

- $l_{unlabeled}$ - the total length of states in a path that do not produce a symbol
- $l_{badLabel}$ - the total length of states in a path that produce an undesired label
- $l_{worstPassThrough}$ - the total length of states in a path that produce a negated label

Indicator functions for termination:

- 1_{goal} - 1 if terminates at goal, 0 otherwise
- $1_{badTerm}$ - 1 if terminates at bad state, 0 otherwise
- $1_{worstTerm}$ - 1 if terminates at negated state, 0 otherwise

Note that the indicator functions are mutually exclusive. That is, only one of them can equal 1. Further, let $l_{max} := l_{unlabeled} + l_{badLabel} + l_{worstPassThrough}$. This is the path length and by definition $\eta \geq l_{max}$ for any *optimal* path.

The total undiscounted reward received for a path between two states, s_1 and s_2 is the following:

$$\begin{aligned} R = R_{goal}1_{goal} + R_{step}* \\ \big[\big(\eta^2 1_{badTerm} + \eta^3 1_{worstTerm} + \eta^4 1_{neverTerm}\big) \\ + \big(l_{unlabeled} + \eta l_{badLabel} + \eta^2 l_{worstPassThrough}\big)\big] \,, \end{aligned} \tag{4}$$

where the first term is the reward for terminating at the goal, the second line consists of penalties for terminating elsewhere, and the third line consists of penalties for passing through other regions.

Since $R_{step} < 0$, the first line is always greater than the second two lines for any path, since it is always non-negative.

We must prove two properties:

1. If a pure path exists, it will be taken, otherwise a minimally violating path will be taken; and
2. If a goal is reachable by *any* path, the MV path will be taken, without terminating early at an undesired goal.

If a pure path exists, then 1_{goal} is achievable and there exists a path consisting entirely of $l_{unlabeled}$. Since $l_{max} \geq l_{unlabeled}$ and $R_{step} \leq 0$, we know that

$$\begin{aligned} R_{pure} &= R_{goal} + R_{step} l_{unlabeled} && (5)\\ &\geq R_{goal} + R_{step} l_{max} && (6)\\ &\geq R_{goal} + R_{step}\eta \,. && (7) \end{aligned}$$

The last line is equivalent to the total reward for a path of length one, consisting only of a bad label. This implies that a pure path always returns a higher reward than a path with even a single bad label. This is because η exceeds l_{max}. Therefore, the same logic holds for states that pass through negated goals or achieve other pass-through penalties, due to the increase of each penalty by η.

Remark 3.2. Note that we assume $\eta \geq l_{max}$. While this is necessary in theory, in practice it is typically sufficient to use a penalty of relatively large magnitude. Relaxing this assumption in practice is useful for speeding up convergence.

In the worst case, the MV path reaches goal g via a path that achieves a penalty for a bad step at every step. The total accumulated reward for such a path is

$$\begin{aligned} R_{worstMinViol} &= R_{goal} + R_{step}\eta l_{badLabel} && (8)\\ &\geq R_{goal} + R_{step}\eta l_{max} && (9)\\ &\geq R_{goal} + R_{step}\eta^2 \,, && (10) \end{aligned}$$

while the reward for a path of length 1 that terminates at any other goal is

$$\begin{aligned} R_{earlyTerm} &= R_{step}\eta^2 + R_{step} && (11)\\ &\leq R_{step}\eta^2 + R_{goal} && (12)\\ &\leq R_{worstMinViol} \,, && (13) \end{aligned}$$

where the first inequality follows from the fact that $R_{goal} > R_{step}$, and the second follows from (8)-(10). Since a path of length 1 that terminates at any other goal is worse than the worst-case MV path, any longer path that terminates at any other goal is also worse than the longest MV path. □

Theorem 2. The reward structure in (2) for positive tasks and (3) for negated tasks produces PS paths.

Proof. For negated tasks trained with (3), we can follow similar logic as the proof of Theorem 1. Here, a path of length one that passes through a state containing a symbol that violates PS produces a reward of

$$R_{priorSafety} = R_{goal} + R_{step}\eta^2 \quad (14)$$
$$\geq R_{goal} + R_{step}\eta l_{max} \quad (15)$$
$$= R_{goal} + R_{badStep} l_{max}\,, \quad (16)$$

where the second line follows from the fact that $\eta \geq l_{max}$. The last line is equivalent to the reward accrued by a path of length $l_{max} - 1$ that exclusively passes through states that give a reward of $R_{badStep}$. Therefore, any path that passes through a negated label achieves a reward that is worse than the longest possible path that passes through any other (non-goal) labels. Otherwise, the hierarchy follows the same pattern as MV policies. □

4. Policy Composition

This section details how to compose policies trained under the safety semantics introduced in Sec. 3, with the goal of learning several tasks up front and then composing them to create a superexponential number of new tasks with no further training via a Boolean algebra over those tasks.

Let a set of tasks be a collection of MDPs which differ only in reward function, R, and *goal* states, $\mathcal{G} \subseteq \mathcal{S}$. To combine tasks in a Boolean fashion, we use methods first developed in Nangue Tasse et al. (2020). The notion for extended reward functions and extended Q-value functions is reproduced below for completeness (see Nangue Tasse et al. (2020) for more detail).

Definition 6. An extended Q-value function $\bar{Q} : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \to \mathbb{R}$ is defined

$$\bar{Q}(s, g, a) = \bar{r}(s, g, a) + \sum_{s' \in \mathcal{S}} \tau(s, a, s')\bar{V}^{\bar{\pi}}(s', g)\,, \quad (17)$$

where $\bar{V}^{\bar{\pi}}(s', g)$ is the value function corresponding to policy $\bar{\pi}$ under reward $\bar{r}$.

For a given task, there is a corresponding MDP with its own extended reward, as well as an associated extended Q-value function. The policy $\bar{\pi}$ corresponding to an extended Q-value function is obtained by taking the $\max$ over $g \in \mathcal{G}$. By explicitly including goals as inputs, extended Q-value functions maintain knowledge of optimal values when the goal corresponds to the task encoded in the extended reward, as well as when the goal does not correspond to the desired task. Intuitively, this captures how good or bad a given goal is in relation to the task. This concept is crucial for the correct composition of policies.

The Boolean operations of conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$) can be performed over tasks in a zero-shot fashion as follows:

$$\bar{Q}^*_1 \vee \bar{Q}^*_2(s, g, a) = \max\{\bar{Q}^*_1(s, g, a), \bar{Q}^*_2(s, g, a)\}\,, \quad (18)$$
$$\bar{Q}^*_1 \wedge \bar{Q}^*_2(s, g, a) = \min\{\bar{Q}^*_1(s, g, a), \bar{Q}^*_2(s, g, a)\} \quad (19)$$
$$\neg\bar{Q}^*(s, g, a) = (\bar{Q}^*_{\mathcal{U}}(s, g, a) + \bar{Q}^*_{\varnothing}(s, g, a)) - \bar{Q}^*(s, g, a) \quad (20)$$

where $\bar{Q}^*_{\mathcal{U}}$ and $\bar{Q}^*_{\varnothing}$ correspond to the extended Q-value functions for the max and min over all rewards, respectively. Further details are in Nangue Tasse et al. (2020) as well as Sec. C in the Supplementary Material.

Prioritized Safety Composition. The PS reward structure of Equation 3 produces the correct semantics for a single negated task. Often, this works for composition as well. However, due to some edge cases, we cannot guarantee that (non-pure) PS paths will be taken under arbitrary compositions of negated tasks. Intuitively, this is because policies encode optimal paths for each goal, but the optimal PS path for combinations of negated tasks may not correspond to an optimal path for any of the original policies. This can result in chattering (infinite loops) in some cases. To provide formal guarantees we must make an additional assumption.

Assumption 4.1. We assume either: i) only a single learned negated policy is used in composition at a time during deployment [1]; or, ii) any PS path in the environment of interest will only have to pass through a known, finite number, k, of non-satisfying regions in $\mathcal{G}$ and we train an extended value function that maintains more corresponding paths.

Intuitively, the issue is that different negated tasks may not agree on the same path to a given goal, since they must avoid different symbols. This can lead to chattering in some cases. Assumption 4.1 manages this problem by limiting the usage to one negated policy at a time, or adding bookkeeping to maintain more paths for avoiding certain (combinations of) symbols. See the Supplementary Details for discussion of this assumption, and details on how to train extended value functions that maintain more paths.

4.1. Theoretical Analysis

Our contribution is proof that composition rules (18)–(20) respect the safety semantics defined in Sec. 2 when trained using the rewards defined in Sec. 3.

Assumption 4.2. For each task p, there exists an optimal policy π_p, and associated Q- and value-functions, Q_p and V_p, respectively. We are agnostic to how these policies are produced. I.e., π_p can be found via dynamic programming (such as policy or value iteration), or by deep RL techniques.

Assumption 4.2 means we can focus on how to apply the results that learning a task provides to us, rather than the

[1] This policy can be more complex than a single task (e.g., one could learn $\neg p_1 \wedge \neg p_2$), but it must be learned ahead of time.

learning process itself. We also remind the reader of our assumption that rewards and penalties only differ on labeled states, defined by Defs. (2) and (3), as applicable.

For an extended Q-value function, each tuple (p, g, s) of task, goal, and state contains information about the value of each possible action and therefore the best action to take from state s. Taken in sequence, this induces an MV path for that task-goal pairing.

Remark 4.3. Note that the proof of composition for PS semantics only holds for negation in the case that there is a path free of bad states (e.g. states satisfying ϕ_B from Definition 3). If no such path exists, it is possible to obtain chattering in the policy even under Assumption 4.1.

We can now provide our three main theorems pertaining to safety semantics under composition.

Theorem 3. The disjunction operator defined in (18) results in policies that satisfy the disjunction of their goals, while maintaining the safety semantics of the individual policies.

Theorem 4. The conjunction operator defined in (19) results in policies that satisfy the conjunction of their goals, while maintaining the safety semantics of the original policies.

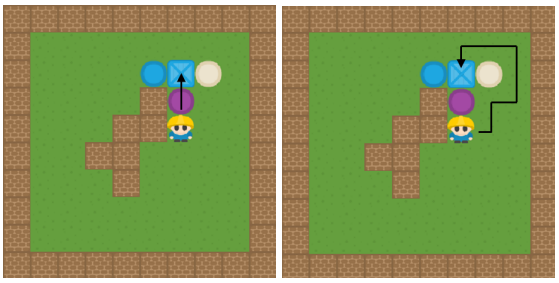
Theorem 5. The negation operator defined in (20) for MV semantics results in policies that correspond to the negation of their original goals and retain MV semantics.

Proof Sketch 1. Intuitively, the intertask rewards respect the intratrask hierarchical reward we introduced in Sec. 3. That is, if a path is safe for one task, the semantics of composition will not alter the hierarchical rewards within a task, nor will it induce other behavior, such as chattering. Complete proofs are in Appendix B.

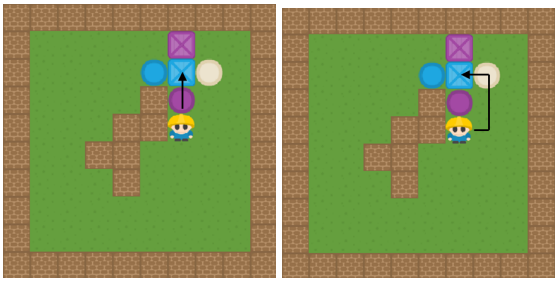
Policies composed with these semantics will produce pure paths when possible, and MV or PS paths if necessary. The resulting traces behave differently than those produced by the RO semantics of Nangue Tasse et al. (2020). Fig. 2 depicts a comparison between the two. Generally, RO semantics will pass through any regions (labeled or not) that fall on the shortest path to a goal state.

There are trade-offs associated with the choice of safety semantics. Namely, MV safety is a weaker notion of safety, treating all non-satisfying states as equally undesirable. In exchange for that weaker notion of safety, the negation operation defined in (20) maintains the MV semantics. On the other hand, PS semantics have a stronger notion of safety, enforcing strict avoidance of a certain subset of $\mathcal{G}$. To accomplish these semantics, we require Assumption 4.1 and must train negated policies separately.

See the Appendix for additional explanation and experiments highlighting the fundamental challenge of PS and the proposed solutions based on Assumption 4.1.



(a) RO: *square* (b) SA: *square*



(c) RO: $square \wedge \neg purple$ (d) SA: $square \wedge not\text{-}purple$

Figure 2: This figure compares reachability-only (RO) semantics to safety-aware (SA) semantics.

5. Continuous Action Spaces

Given two optimal extended Q-value functions, $\bar{Q}_1^*$ and $\bar{Q}_2^*$, with their associated optimal policies, $\bar{\pi}_1^*$ and $\bar{\pi}_2^*$, we can compute the policy corresponding to their conjunction as

$$\bar{\pi}^*_{\bar{Q}_1^* \wedge \bar{Q}_2^*}(s,g) = \begin{cases} \bar{\pi}_1^*(s,g) & \text{if } \bar{Q}_1^*(s,g,\bar{\pi}_1^*(s,g)) \leq \\ & \bar{Q}_2^*(s,g,\bar{\pi}_2^*(s,g)) \\ \bar{\pi}_2^*(s,g) & \text{otherwise .} \end{cases} \tag{21}$$

Similarly, for disjunction, we can compute the resulting policy as

$$\bar{\pi}^*_{\bar{Q}_1^* \vee \bar{Q}_2^*}(s,g) = \begin{cases} \bar{\pi}_1^*(s,g) & \text{if } \bar{Q}_1^*(s,g,\bar{\pi}_1^*(s,g)) \geq \\ & \bar{Q}_2^*(s,g,\bar{\pi}_2^*(s,g)) \\ \bar{\pi}_2^*(s,g) & \text{otherwise .} \end{cases} \tag{22}$$

Negation cannot be computed directly, so we must learn a policy for each negated task. We train negated policies with MV semantics and apply them using NNF.

Theorem 6. Eqs. (21)–(22) facilitate MV task composition.

Proof. To extend Boolean task composition to continuous action spaces for MV semantics, we train negated policies separately. We now prove that disjunction and conjunction over these positive and negated learned tasks behave as expected.

Disjunction For disjunction of two tasks, the associated extended $\bar{Q}$-function (18) requires evaluating the $\max$ oper-

ator. For discrete action spaces, the max can be determined by comparing over all actions in $\mathcal{A}$. For continuous action spaces, that is not feasible, so we must determine the optimal choice analytically.

From a policy standpoint, let's consider $\bar{Q}_1^* \vee \bar{Q}_2^*$. The optimal action will be $\pi^*_{\bar{Q}_1^* \vee \bar{Q}_2^*} : S \to \mathcal{A}$ and is given by

$$\pi^*_{Q_1^* \vee Q_2^*}(s) = \arg\max_{a \in \mathcal{A}} [\max_{g \in \mathcal{G}} \bar{Q}_1^* \vee \bar{Q}_2^*(s, g, a)]$$

where

$$\bar{Q}_1^* \vee \bar{Q}_2^*(s, g, a) = \max\{\bar{Q}_1^*(s, g, a), \bar{Q}_2^*(s, g, a)\} . \tag{23}$$

Note that we can swap the order of maximizing over $a \in \mathcal{A}$ and $g \in \mathcal{G}$. Thus, we denote a policy $\pi(s, g)$ as the optimal action for reaching goal g with the minimum number of penalties. Since we know by definition that $\bar{Q}^*(s, g, \cdot)$ is maximized by the corresponding $\pi^*(s, g)$, then the solution to (23) is either $\pi_1^*(s)$ or $\pi_2^*(s)$ where $\pi_i^*(s) := max_{g \in \mathcal{G}} \pi_i^*(s, g)$. That is, the global maximum for $\bar{Q}_1^* \vee \bar{Q}_2^*(s, g, \cdot)$ is either the global maximum of $\bar{Q}_1^*(s, g, \cdot)$ or the global maximum of $\bar{Q}_2^*(s, g, \cdot)$.

Therefore, the optimal policy for disjunction of two tasks is

$$\pi^*_{\bar{Q}_1^* \vee \bar{Q}_2^*}(s, g) = \begin{cases} \pi_1^*(s, g) & \text{if } \bar{Q}_1^*(s, g, \pi_1^*(s, g)) \geq \\ & \bar{Q}_2^*(s, g, \pi_2^*(s, g)) \\ \pi_2^*(s, g) & \text{otherwise} . \end{cases} \tag{24}$$

The optimal action is chosen using $max_{g \in \mathcal{G}} \pi^*_{\bar{Q}_1^* \vee \bar{Q}_2^*}(s, g)$.

Proof Sketch 2. The proof of correctness follows the same reasoning as Proof B.1.

Conjunction We follow similar logic to reason about conjunction. For conjunction of two tasks, the associated extended $\bar{Q}$-function is given by (19). The best action for a conjunction of terms is written

$$\pi^*_{Q_1^* \wedge Q_2^*}(s) = \arg\max_{a \in \mathcal{A}} [\max_{g \in \mathcal{G}} \min\{\bar{Q}_1^*(s, g, a), \bar{Q}_2^*(s, g, a)\}]. \tag{25}$$

The resulting policy is therefore

$$\pi^*_{\bar{Q}_1^* \wedge \bar{Q}_2^*}(s, g) = \begin{cases} \pi_1^*(s, g) & \text{if } \bar{Q}_1^*(s, g, \pi_1^*(s, g)) \leq \\ & \bar{Q}_2^*(s, g, \pi_2^*(s, g)) \\ \pi_2^*(s, g) & \text{otherwise} . \end{cases} \tag{26}$$

Proof Sketch 3. For MV semantics or PS semantics with Assumption 4.1, we are guaranteed that the composed policies either agree on the value of a path for the same satisfying goal with a high value, or is dominated by a low-value (due to a non-satisfying goal, unsafe action, or long path) due to the minimum operation. Note that there could be more than one optimal path and thus different possible actions. However, for goals that satisfy both tasks, they must agree on the value and thus we do not have a preference between these two paths. Thus, the optimal action is encoded by either π_1^* or π_2^*. The justification follows the same reasoning as Proof B.2.

Note, we face the same limitations for PS in continuous action spaces as in discrete action spaces. However, we can rely on the same assumptions in Assumption 4.1 and leverage the same approaches used in Appendix F to lift this notion to continuous action spaces as well. □

6. Experimental Details and Analysis

We demonstrate safety-aware task composition in three environments:

1. a 2D static grid world with row and column observation spaces and 5 actions (each direction and stay); optimal policies obtained with value iteration

2. a 2D item collection grid world with image observations and 4 actions (each direction); optimal policies approximated by DQN

3. `Bullet-Safety-Gym` (Gronauer, 2022), a 3D physics simulation with 96D LIDAR-like observations and a (continuous) 2D force vector action space; optimal policies approximated by TD3

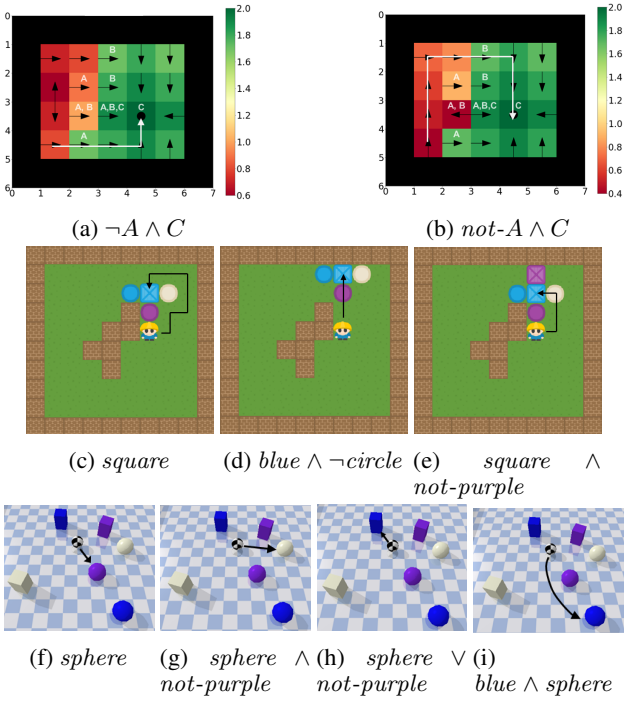
For each experiment, we trained a policy for each base task. For PS semantics, we also trained policies for negated tasks. All Boolean combinations of tasks shown in the results were obtained at run time, by applying (18)-(20) (or (21) and (22) in the continuous case) on the corresponding policies without any further training. All function approximation experiments were conducted with an NVIDIA Volta GPU, and tuned over three learning rates using curriculum learning, where penalties were added after the policy could successfully reach goals. We selected the best policies for the demonstrations over four random seeds. See the Supplementary Material for videos.

For Environment 1, optimal policies were found using value iteration.

For Environment 2, we use a modified version of the code from Nangue Tasse et al. (2020). We made updates to include penalties and perform curriculum learning in the following order:

1. train a model to reach uniformly sampled goal items without any penalties (or use a pre-trained model from Nangue Tasse et al. (2020)); allocated 2M steps, but converges earlier (1M steps)

2. refine the model by further training on random environments but with penalties required for safety added;



(a) $\neg A \wedge C$ (b) $not\text{-}A \wedge C$

(c) $square$ (d) $blue \wedge \neg circle$ (e) $square \wedge not\text{-}purple$

(f) $sphere$ (g) $sphere \wedge not\text{-}purple$ (h) $sphere \vee not\text{-}purple$ (i) $blue \wedge sphere$

Figure 3: Example optimal policies and trajectories. **Environment 1:** Color scale shows the value at the state, arrows show policy direction, and a circle denotes a stay action. 3a shows an MV path using analytical negation and 3b shows a PS path using a learned "not-A" policy. Both have multiple pure paths. Note that these align with the example paths used in Fig. 1c. **Environment 2:** Fig. 3c shows a pure path, 3d shows MV using a negated task, and 3e shows a PS path using a learned negated task, "not-$purple$". **Environment 3:** Fig. 3f depicts a path to the nearest sphere, 3g goes to the nearest sphere that is not purple, 3h goes to the nearest object that is either a sphere or not purple (the blue box), and 3i heads to the blue sphere while avoiding other goals.

allocated 2M steps, but also converges earlier (<1M steps)

3. refine the model further on closely spaced items, because the model learns to perform well on random environments, but those tend to have spread out items rather than tightly constrained layouts; 20-60K steps

For Environment 3 we use a modified version of the TD3 code from Fujimoto et al. (2018). We added a collection task to `Bullet-Safety-Gym` for this environment that mirrored the one in Environment 2. Unlike the previous two environments, we train a dedicated policy for each goal, which we found to perform better in this case. We trained on random environments for up to 4M steps. To achieve the policies used in the demo, we trained on that static environment for an additional 1M steps. We incorporated penalties immediately for this environment, because there did not appear to be any advantage to curriculum learning in this case.

Figs. 3a–3b depict composed optimal policies learned with value iteration for Environment 1 that highlight the different safety semantics. Similarly, Figs. 3c–3e demonstrate different safety semantics in Environment 2. Penalty-free Boolean task composition (Nangue Tasse et al., 2020) heads straight toward a satisfying item without regard for other items in the way (not pictured). Finally, Figs. 3f–3i depict our approach in Environment 3. To our knowledge, this is the first application of Boolean task composition in a continuous action space. See Appendix D for additional analysis and comparisons.

7. Conclusion and future work

We have extended the theory of Boolean task composition in RL to facilitate two notions of safety constraints and support continuous action spaces. We proved correctness of the approach for optimal policies in deterministic MDPs, and demonstrated that it generalizes well to scenarios requiring function approximation. Despite some limitations (See Appendix A), we believe that the general approach of Boolean task composition introduced by Nangue Tasse et al. (2020) is significant and promising. We have addressed two such limitations by introducing safety semantics and continuous action spaces support, and recent work has demonstrated the ability to extend composition to stochastic and discounted settings (Nangue Tasse et al., 2022a). Future work can take composition even further, including (potential-based) reward shaping, reducing redundancy in learning the extended value functions, and new techniques for solving the planning problem to determine which Boolean compositions to execute for more complex sequences of tasks (e.g., as defined by a temporal logic).

Impact statement

This paper presents work whose goal is to advance the field of Reinforcement Learning (RL). There are many potential societal consequences of RL, and Machine Learning more broadly. We hope that this work facilitates use of RL that is more transparent and comprehensible to users, facilitating greater trust and reliability in RL.

Acknowledments and disclosure of funding

Thank you to the anonymous reviewers and Paula Donovan for the their thoughtful feedback.

The NASA University Leadership initiative (grant #80NSSC20M0163) provided funds to assist the authors with their research, but this article solely reflects the opinions and conclusions of its authors and not any NASA entity.

References

Achiam, J., Held, D., Tamar, A., and Abbeel, P. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 22–31. PMLR, 06–11 Aug 2017.

Adamczyk, J., Tiomkin, S., and Kulkarni, R. Leveraging prior knowledge in reinforcement learning via double-sided bounds on the value function, 2023.

Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., and Topcu, U. Safe reinforcement learning via shielding. In *Proceedings of the AAAI conference on artificial intelligence*. AAAI Press, 2018. ISBN 978-1-57735-800-8.

Ames, A. D., Coogan, S., Egerstedt, M., Notomista, G., Sreenath, K., and Tabuada, P. Control barrier functions: Theory and applications. In *2019 18th European Control Conference (ECC)*, pp. 3420–3431, 2019. doi: 10.23919/ECC.2019.8796030.

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., and Mané, D. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

Baier, C. and Katoen, J. *Principles of model checking*. MIT Press, 2008.

Balakrishnan, A., Jakšić, S., Aguilar, E. A., Ničković, D., and Deshmukh, J. V. Model-free reinforcement learning for symbolic automata-encoded objectives. *arXiv preprint arXiv:2202.02404*, 2022.

Belta, C., Yordanov, B., and Gol, E. *Formal Methods for Discrete-Time Dynamical Systems*, volume 89. Springer, 01 2017. ISBN 978-3-319-50762-0. doi: 10.1007/978-3-319-50763-7.

Berducci, L., Aguilar, E. A., Ničković, D., and Grosu, R. Hierarchical potential-based reward shaping from task specifications. *arXiv preprint arXiv:2110.02792*, 2021.

Dawson, C., Gao, S., and Fan, C. Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods for robotics and control. *IEEE Transactions on Robotics*, pp. 1–19, 2023. doi: 10.1109/TRO.2022.3232542.

Fujimoto, S., Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

Gronauer, S. Bullet-safety-gym: A framework for constrained reinforcement learning. Technical report, mediaTUM, 2022.

Ibarz, J., Tan, J., Finn, C., Kalakrishnan, M., Pastor, P., and Levine, S. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40(4-5):698–721, 2021.

Jothimurugan, K., Alur, R., and Bastani, O. A composable specification language for reinforcement learning tasks. *Advances in Neural Information Processing Systems*, 32, 2019.

Jothimurugan, K., Bansal, S., Bastani, O., and Alur, R. Compositional reinforcement learning from logical specifications. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W. (eds.), *Advances in Neural Information Processing Systems*, 2021.

Nangue Tasse, G., James, S. D., and Rosman, B. A boolean task algebra for reinforcement learning. In *NeurIPS*, 2020.

Nangue Tasse, G., James, S., and Rosman, B. Generalisation in lifelong reinforcement learning through logical composition. In *International Conference on Learning Representations*, 2022a.

Nangue Tasse, G., Jarvis, D., James, S., and Rosman, B. Skill machines: Temporal logic composition in reinforcement learning. *CoRR*, abs/2205.12532, 2022b.

Ng, A. Y., Harada, D., and Russell, S. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, pp. 278–287. Morgan Kaufmann, 1999.

Pnueli, A. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pp. 46–57, 1977. doi: 10.1109/SFCS.1977.32.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609, 2020.

Taylor, M. E. and Stone, P. Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.*, 10:1633–1685, dec 2009. ISSN 1532-4435.

Tessler, C., Mankowitz, D. J., and Mannor, S. Reward constrained policy optimization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SkfrvsA9FX.

Yang, T.-Y., Rosca, J., Narasimhan, K., and Ramadge, P. J. Projection-based constrained policy optimization. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rke3TJrtPS.

Žikelić, Đ., Lechner, M., Verma, A., Chatterjee, K., and Henzinger, T. A. Compositional policy learning in stochastic control systems with formal guarantees. *arXiv preprint arXiv:2312.01456*, 2023.

A. Limitations

We inherit many of the same limitations of Nangue Tasse et al. (2020), including a sparse reward structure and reliance on deterministic MDPs for theoretical guarantees. We expect that reward shaping (Balakrishnan et al., 2022; Berducci et al., 2021; Ng et al., 1999) can be adapted to this scenario to address the former, and the latter can easily be relaxed in practice. Further, Boolean task composition approaches for RL depend on the MDP being identical across tasks except the reward, and, for certain environments, not all Boolean compositions may be valid. In practice, we observed that penalties for the safety semantics made it more difficult for policies to converge. This is expected because reachability under safety constraints is complex and may require greater global reasoning to find pure or minimally violating paths. In addition, the rewards start quite negative in early exploration. We addressed this via curriculum learning, but we expect that future work in reward shaping can better address this limitation. We also note that the modularity of Boolean task composition helps identify issues during training, because each individual task can be inspected independently and trained for longer as needed.

B. Proofs

For the following proofs, we introduce Lemma B.1 below.

Lemma B.1. *From a given state s, a policy $\bar{\pi}$ trained under the reward structure given by (2) (resp. (3)) encodes a path corresponding to the shortest MV (resp. PS) path from s.*

Proof Sketch 4. We assume a deterministic MDP. Then, using the definitions of optimal policies, this lemma follows directly from Theorems 1 & 2, respectively.

B.1. Proof of Theorem 3

Proof. For two extended Q-value functions, $\bar{Q}_1$ and $\bar{Q}_2$, disjunction defined by (18) results in policy

$$\bar{\pi}_{1\vee 2}(s) = \arg\max_{a\in\mathcal{A}}\{\max_{g\in\mathcal{G}} \max\{\bar{Q}_1(s,g,a), \bar{Q}_2(s,g,a)\}\} \,. \tag{27}$$

For policies $\bar{\pi}_1$ and $\bar{\pi}_2$ that each encode MV semantics (resp. PS semantics) and an arbitrary state s, we can assume without loss of generality that $\bar{\pi}_1$ encodes the shorter MV path to some goal g. Note that $\bar{\pi}_2$ may encode a path for either the same goal g or a different goal g'. If we call their corresponding executions x_1 and x_2, then we know (by definition) that $|\restriction^+_L(x_1)| \leq |\restriction^+_L(x_2)|$, so the sequence of labeled states must be shorter for x_1 as well. Since they are both MV, and all rewards other than the goal reward are negative, the shorter path has a higher cumulative reward-to-go, and therefore $\bar{V}_1(s) \geq \bar{V}_2(s)$. Because $\bar{\pi}$ encodes an optimal path for at least one goal g, the same reasoning can be applied for s', the state reached from s by applying $\bar{\pi}(s)$. This is equivalent to the definition of disjunction, and therefore respects MV semantics (resp. PS semantics). □

Assumption B.2. For the following proof, we assume that conjunction is semantically meaningful. I.e., for $p_1 \wedge p_2$, there exists at least one goal that satisfies $p_1 \wedge p_2$.

B.2. Proof of Theorem 4

Proof. For two extended Q-value functions, $\bar{Q}_1$ and $\bar{Q}_2$, conjunction defined by (19) results in policy

$$\bar{\pi}_{1\wedge 2}(s) = \arg\max_{a\in\mathcal{A}}\{\max_{g\in\mathcal{G}} \min\{\bar{Q}_1(s,g,a), \bar{Q}_2(s,g,a)\}\} \,. \tag{28}$$

Consider tasks p_1 and p_2, and their corresponding optimal extended value functions, $\bar{Q}^*_1$ and $\bar{Q}^*_2$. Let $g^* \in \mathcal{G}$ be the goal that satisfies both tasks with the lowest-penalty path from s. Further, let $g \in \mathcal{G}$ be a goal that satisfies both tasks but has no lower penalty path from s than g^*, and $g' \in \mathcal{G}$ be a goal that does not satisfy either of the tasks.

For simplicity, we first consider MV semantics. By construction, $max_{a\in\mathcal{A}}\bar{Q}^*_1(s,g^*,a) = max_{a\in\mathcal{A}}\bar{Q}^*_2(s,g^*,a)$ because g^* satisfies both tasks and the penalties are identical in MV. Thus, the maximizing action is identical for both value functions. There can be more than one maximizing action, in which case we have no preference.

Furthermore, we know that $max_{g\in\mathcal{G},a\in\mathcal{A}}\bar{Q}^*_i(s,g,a) < max_{g\in\mathcal{G},a\in\mathcal{A}}\bar{Q}^*_i(s,g^*,a)$ because there is no lower penalty path to g. Applying the minimum operator for conjunction preserves this element-wise inequality and thus, the composed policy will choose actions for the path to g^*.

Finally, because the $\eta^2 R_{step}$ penalty on bad terminations, either $\bar{Q}_1^*(s, g', a)$ or $\bar{Q}_2^*(s, g', a)$ has a low value compared to the paths to g^*. Due to the minimum operation in conjunction, this is the preserved value for $\bar{Q}_{1\wedge 2}^*$ and $max_{g^*\in\mathcal{G}, a\in\mathcal{A}}\bar{Q}_{1\wedge 2}^*(s, g, a) < max_{g^*\in\mathcal{G}, a\in\mathcal{A}}\bar{Q}_{1\wedge 2}^*(s, g^*, a)$. Thus, the composed policy will choose actions corresponding to paths to g^*.

For PS semantics, we rely on Assumption 4.1. Using Option 1 from the assumption, we have only a single conjuncted safety constraint. Unsafe actions leading to $R_{worststep}$ penalties have low values that dominate the minimum operation. The termination penalties are the same. The chosen path will correspond to the best PS path to a goal that satisfies both the safety constraint and the other conjunct. See Appendix F for more details and an explanation of Option 2. □

B.3. Proof of Theorem 5

Proof. Following Lemma B.1, let $\tau_{s,g}$ denote the shortest MV path between a state s and a goal g, not including g, with associated reward $r_{s,g}$. Considering MV semantics, when a task is negated, the reward only varies on states where `done` is true. All step rewards (good or bad) are the same. Therefore, two paths vary only on their terminal cost between a task and the negation of a task. Then, training a negated task directly corresponds to

$$\bar{Q}_{\neg p}^*(s, g, a) = r_{s,g} + R_{\neg p}\ , \tag{29}$$

where $R_{\neg p}$ is the cost of bad termination, i.e., $R_{badterm} = \eta^2 R_{step}$.

The negation operator results in extended value function

$$\neg\bar{Q}_p^*(s, g, a) = r_{s,g} + \neg R_p\ , \tag{30}$$

where $\neg R_p$ is given by $(r_{MAX} + r_{MIN}) - R_{goal}$. Since r_{MAX} equals R_{goal} and r_{MIN} equals $R_{badterm}$ (See Appendix C) then (29) equals (30). □

C. Maximum and minimum policy

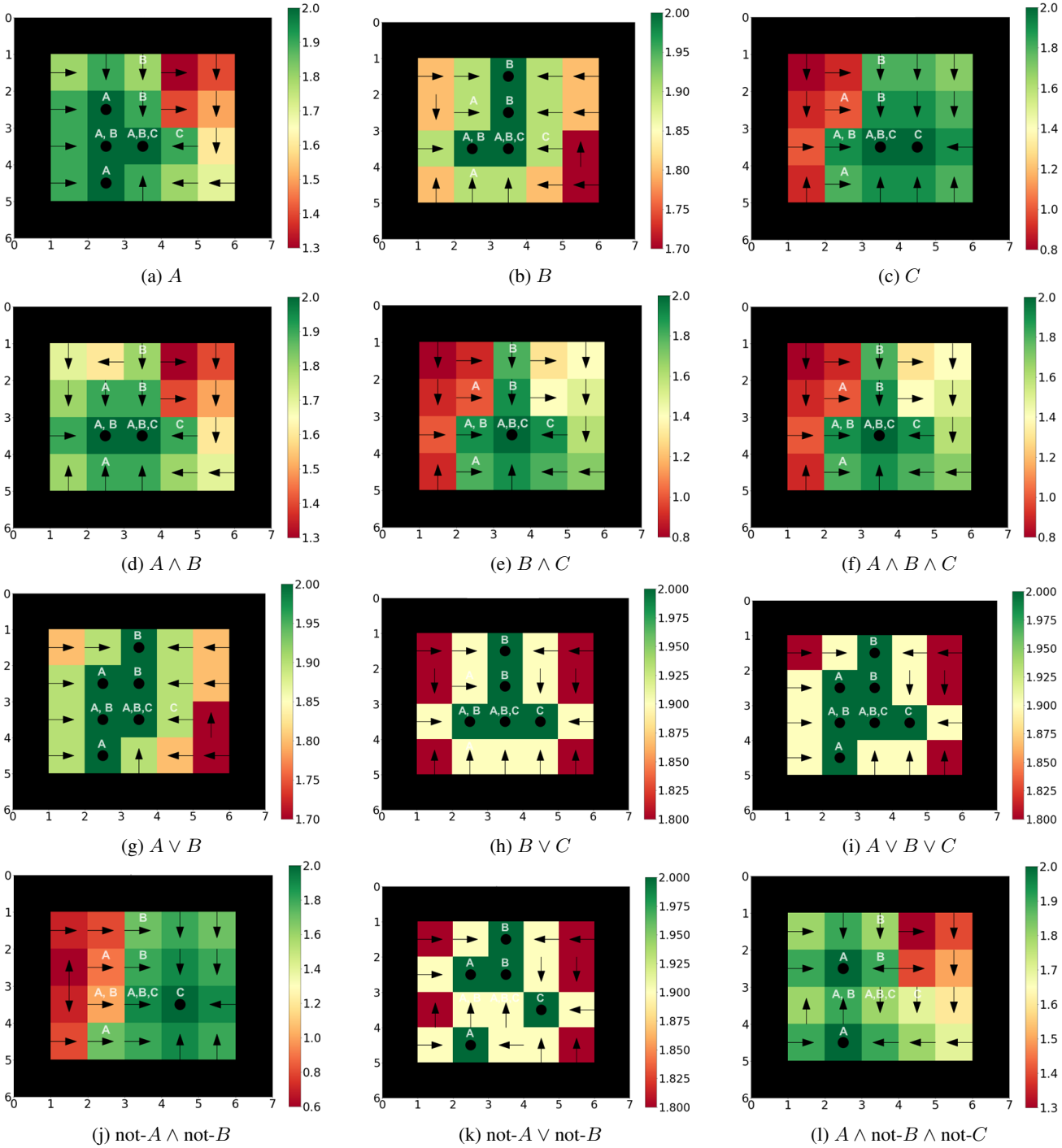
For MV semantics and the semantics of Nangue Tasse et al. (2020), there is a minimum policy used in negation, $\bar{Q}_{\mathcal{U}}^*$ and $\bar{Q}_{\varnothing}^*$. See Sec. 4 for a review. These policies can either be approximated, or learned. To learn the maximum policy, give the goal reward for every goal in G. For the minimum policy, give the worst expected penalty for each goal in G. Note that this is not the penalty used to prevent terminating at the wrong goal, which should never occur assuming the environment contains that goal. In Nangue Tasse et al. (2020) this is the step penalty. In our case, it is the bad termination penalty, $R_{badterm}$.

D. Additional Experimental Results

In this section we expand on the experimental results of Sec. 6. Fig. 4 depicts several other example compositions in the simple grid world. Recall that these policies are optimal because this is converged value iteration and has no function approximation. The negated policies in this figure are learned for PS (as opposed to the MV semantics given by the negation operator in (20)). We did not utilize Assumption 4.1 and the associated approaches detailed in F. This demonstrates that PS often works without chattering in practice, despite requiring additional assumptions for a formal guarantee.

E. Approximated Training Steps to Achieve All Tasks

Fig. 5 shows a comparison of approximate training steps required for different approaches to achieve all Boolean combinations of an increasing number of tasks. We considered a policy converged when a rolling average of evaluated rewards settles within 3% of the overall maximum reward during training in the discrete and continuous environments. We approximately extrapolate the training steps for each approach by multiplying the number of policies that need to be learned for different approaches by the time to train one task or the combination of base tasks. We compare the (approximated) total times to achieve all tasks for: i) learning extended value functions for positive tasks only (using analytic negation), for composition with MV; ii) learning extended value functions for positive tasks and all possible safety constraints (Option 1 of Assumption 4.1), for composition with PS; iii) learning all Boolean combinations (regular value functions) directly for positive tasks only, which we call individual tasks with MV; and iv) learning all Boolean combinations (regular value functions) directly for positive and negated tasks, which we call individual tasks with PS. This plot is on a log scale, but the difference is still exponential because the number of possible compositions is doubly exponential in the individual task combination traning



(a) A (b) B (c) C

(d) $A \wedge B$ (e) $B \wedge C$ (f) $A \wedge B \wedge C$

(g) $A \vee B$ (h) $B \vee C$ (i) $A \vee B \vee C$

(j) not-$A \wedge$ not-B (k) not-$A \vee$ not-B (l) $A \wedge$ not-$B \wedge$ not-C

Figure 4: Several task composition examples in the example environment using value iteration.

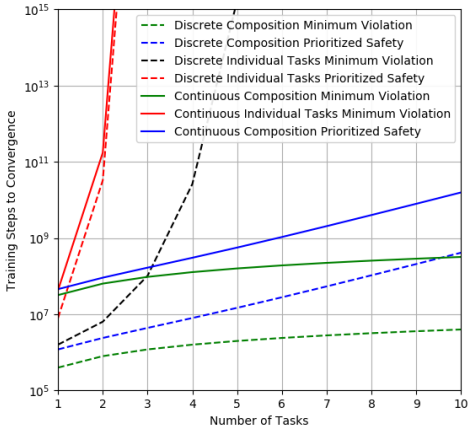


Figure 5: Comparison of approximated training steps required for each method

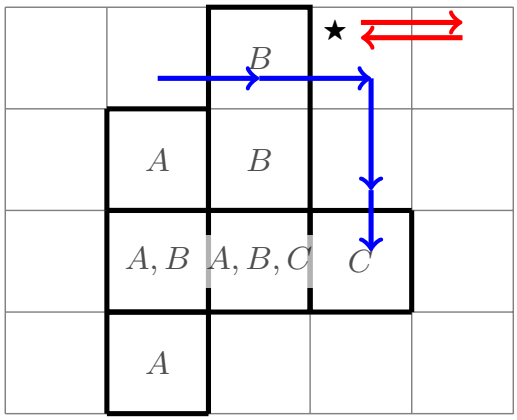


Figure 6: Example optimal vs non-optimal path

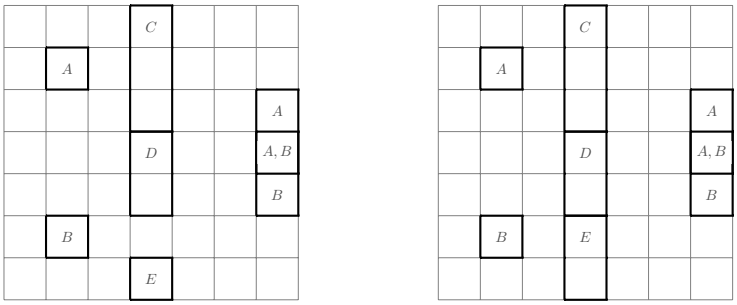
cases. We observe that the cost of learning negated tasks for PS is negligible at this scale. Furthermore, even learning all possible safety constraints scales significantly better than learning all individual tasks. We also note that many use cases do not require learning all possible safety constraints. The environment of interest may only have a subset of the space of safety constraints that is relevant. Also, note that moving from the discrete to continuous domain, there is an increase in training time within a given approach, but that overall increase in training time is negligible in comparison to the individual learning cases as the number of tasks considered grows.

F. Prioritized safety assumption and solutions

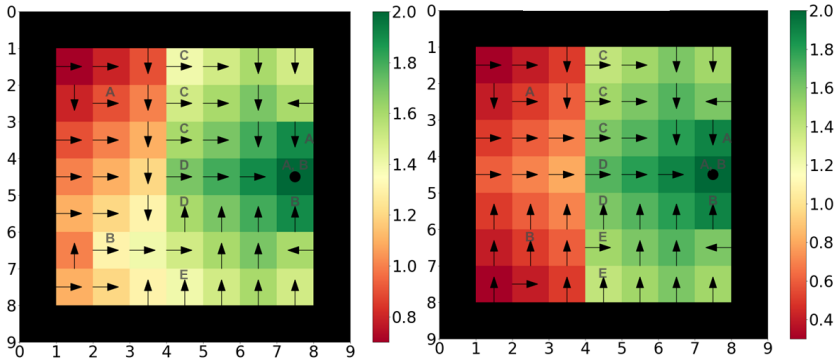
Here we provide more information on Assumption 4.1 of Section 3.1. Recall that PS puts extra weight on avoiding specifically negated region labels and that we train negated tasks explicitly. Due to the avoidance assymmetry, we cannot guarantee that composition between negated tasks works as desired in every case (despite it often working practice) without an additional assumption. The primary failure mode is chattering (infinite loops).

Chattering occurs when the optimal policies for negated tasks have not encoded the same pure paths and the environment layout causes them to disagree. In this case, the best action in the composed policy might not be the optimal action from either negated task. Note that only optimal paths are stored in an extended value function. Any non-optimal action typically reflects the value of moving off and back onto an optimal path.

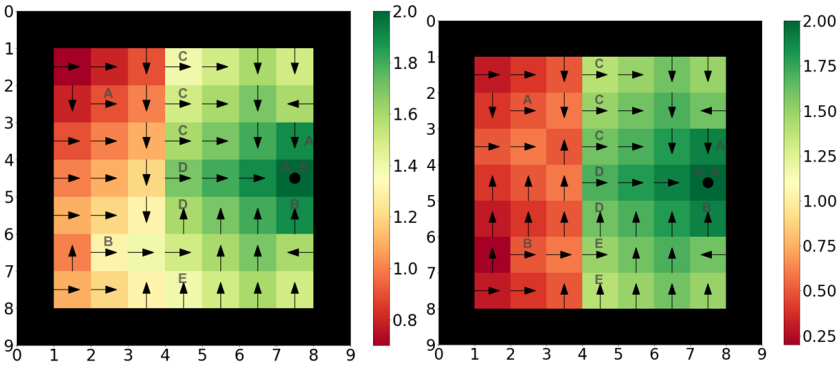
For example, consider the optimal path depicted in blue in Figure 6. Let the value for action a and goal C at the ★ state be



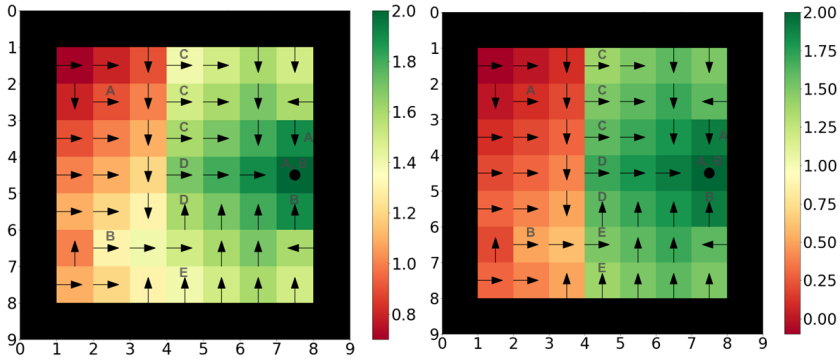
(a) Environments



(b) Minimum violation



(c) Prioritized safety



(d) Prioritized safety (combined $\neg C \wedge \neg D$)

Figure 7: A comparison of semantics when there are multiple negations in environments handpicked to show issues. Every policy is solving this specification: $A \wedge B \wedge \neg C \wedge \neg D$. The left column shows a case where there are pure paths and the right column is a slightly different environment with no pure paths. The combined PS approach learned one policy for $(\neg C \wedge \neg D)$. Note the chattering at (2, 3) in 7c.

denoted $\bar{Q}^*(\bigstar, C, a)$. Then $\bar{Q}^*(\bigstar, C, right) = \bar{Q}^*(\bigstar, C, down) + 2R_{step}$. This is because stepping to the right does not fall on an optimal path and thus the value function is encoding the steps depicted by the red arrows that step away and back onto the optimal path. Depending on the region layout, this can lead to chattering under PS semantics. For the remainder of this section, we focus on conjunctions in NNF because this is how chattering arises.

Assumption 4.1 proposed two options for circumventing this challenge:

1. only a single negated policy is used in composition at a time during deployment [note this policy can be more complex than a single task (e.g., learn $\neg p_1 \wedge \neg p_2$), but it must be learned ahead of time]; or,

2. any PS path in the environment of interest will only have to pass through a known, finite number, k, of non-satisfying regions in $\mathcal{G}$ and we train an extended value function that maintains more corresponding paths.

Option 1 can still represent any Boolean formula, but any safety constraints that may be requested at deployment must be known and trained in advance. This option makes sense in scenarios where potential safety constraints are clear at training time and they can be pre-trained. Note that any safety constraint that must be followed in all cases can trivially be included in the training procedure for every task. Thus, we are concerned with safety constraints that need to be enabled or disabled at deployment time based on user input. Furthermore, because they are composable with reachability policies, this is still a much more efficient re-use of learned policies than explicitly learning every combination.

Option 2 can be composed arbitrarily at deployment but requires learning additional Q-table entries, each of which has a lower penalty for passing through different subsets of $\mathcal{G}$. Let a *safety extended value function* $\bar{Q}(s, g, G_{ok}, a) : \mathcal{S} \times \mathcal{G} \times 2^{\mathcal{G}} \times \mathcal{A} \rightarrow \mathbb{R}$ be an extended Q-value function that behaves identically to an extended Q-value function except that it provides a lighter penalty for passing through any $g_{ok} \in G_{ok}$. This is accomplished by shifting rewards as needed via the η multiplier.

A safety extended value function maintains additional optimal paths for scenarios in which passing through certain goal regions is allowed. Thus, it maintains more paths. The rewards are structured to still prefer pure paths. The assumption in Option 2 states that all required members of G_{ok} are known at training time. This depends on domain knowledge about the environment. It might be that the environment regions are not closely packed and there are only a few scenarios in which passing through a different goal region for PS is required. Those goal regions can be added to G_{ok} to maintain paths that are allowed to pass through those regions at a smaller penalty. With these additional paths, Boolean composition works the same way. Paths that satisfy all the composed tasks have the highest value because they agree. If there is no pure path, then one of the paths passing through an allowed label region may be the best path. Negated tasks will dominate saved paths that pass through violating regions with a very negative value, keeping them from being chosen as the optimal action.

This approach increases the number of required Q-table entries by a factor of k and if $k = 2^{|\mathcal{G}|}$ then it can handle any possible prioritized MV path for any placement of $\mathcal{G}$ regions. This option is a good choice if $k << 2^{|\mathcal{G}|}$. The value of k is based on domain knowledge of the environment of interest. In our experiments, we use Option 1 or relax the assumption to show that it often works in practice without it.

F.1. Semantics Comparison Example

We performed additional experiments in a grid world using value iteration to obtain optimal policies. The results are shown in Fig. 7. All depicted policies are compositions performing $A \wedge B \wedge \neg C \wedge \neg D$. There are two very similar environments. The only difference is whether there are one or two states labeled "E." In the former case, there is a pure path from any state to an $\{A, B\}$ goal state. In the latter, there are some states with no pure path to the goal state. We compare three different semantics on these two environments:

1. MV semantics

2. PS semantics used naively

3. PS semantics using Option 1 to learn a single policy for $(\neg C \wedge \neg D)$

In the pure environment, we expect all policies to be identical. Note, the actions could differ at a particular state if there are multiple shortest pure paths. By contrast, in the non-pure environment, we expect a difference between the policies.

For MV, it only tries to minimize the number of produced symbols that do not satisfy the specification. Thus it will go through the “D" region even though “D" is negated.

For PS applied naively, with the safety constraint formed by composing a policy learned for $\neg C$ with a policy learned for $\neg D$, we see chattering. This is because the two negated policies do not compose properly on this particular environment, which was handpicked to induce chattering. See states $(3, 3)$ and $(3, 2)$.

Finally, to address this issue we show PS using the approach described in Option 1 above. Here we know ahead of time that we might need $\neg C \wedge \neg D$ at deployment and train a dedicated policy for that combination. When we compose this with the reachability policies learned for “A’ and “B", we get the expected behavior. In this case, that is to prioritize avoiding both “C" and “D" which results in taking a path through “E" to get to the final desired state.

Note that in many cases naive compositions of negated PS policies will behave perfectly fine in an environment. However, it is possible to construct difficult environments that induce chattering, thus we cannot formally guarantee correct composition without Assumption 4.1.