Safe and Generalizable Reinforcement Learning via Logical Policy Composition

Xinyang Chen
Imperial College London
London, United Kingdom
xinyangchen4@gmail.com

Francesco Belardinelli
Imperial College London
London, United Kingdom
francesco.belardinelli@imperial.ac.uk

Alex Goodall
Imperial College London
London, United Kingdom
a.goodall22@imperial.ac.uk

ABSTRACT

Reinforcement learning (RL) excels at optimizing policies for narrowly defined tasks, but it often struggles with *safety* and *generalization* in complex environments. Designing a single reward function to encode both goals and safety constraints can lead to unsafe behavior, as agents may exploit poorly shaped rewards. Similarly, standard RL agents tend to overfit to specific tasks, lacking the ability to reuse skills for new goals without retraining. In this work, we propose Safe-Comp, a hierarchical RL framework that addresses both challenges by integrating formally defined goals and compositional skill reuse. Our approach first trains generic primitive policies for fundamental skills under safety-aware rewards. These skills are then composed at a higher level to satisfy arbitrary task specifications expressed in linear temporal logic (LTL). In particular, given a new task (e.g. "achieve goal a while avoiding region b, then go to c"), we translate the corresponding LTL formula into a deterministic finite automaton (DFA) and plan a safe policy by combining the learned primitives according to the DFA structure. This logical composition ensures that safety rules are never violated and enables zero-shot generalization to an extensive variety of novel task combinations, without additional training.

KEYWORDS

Reinforcement Learning, Task Composition, Hierarchical Learning, Skill Generalization, Linear Temporal Logic

1 INTRODUCTION

Reinforcement learning (RL) [33] has achieved impressive successes in domains such as robotics [13, 14, 24], resource management [28], and games [27, 29]. An RL agent learns by interacting with an environment to maximize the expected cumulative reward. In practice, however, standard RL methods face two key limitations when tasks become more complex or safety-critical. First, agents optimized solely for reward may learn unsafe policies that violate important constraints (e.g., collisions or resource exhaustion) if those constraints are not adequately reflected in the reward function. This is a well-known issue: designing a reward that perfectly balances task achievement and safety is often infeasible. As a result, policies may exploit reward loopholes and lead to dangerous behavior [4, 34]. Second, conventional RL agents generalize poorly – a policy trained for one specific goal usually cannot handle even slight variations of that task. The agent must be retrained for each new objective, indicating a lack of skill reuse or transfer. In contrast, an intelligent agent should leverage prior skills to adapt to new tasks with minimal additional learning [40].

A key challenge for safety is that complex tasks often involve multiple objectives and constraints, possibly logical in nature (e.g. reach-avoid goals like "avoid region a until goal b is reached"). A single static reward signal cannot easily capture such conditional or temporal requirements [18]. Hand-tuning scalar penalties for each constraint quickly becomes intractable without a principled way to encode the task's logical structure [25]. Recent work in safe RL has therefore turned to formal methods – for example, by specifying safety conditions in temporal logic [15] and using automata-based monitors or shields to prevent unsafe actions [2, 6]. Linear Temporal Logic (LTL), in particular, provides a rigorous way to express safety requirements and objectives over time. By leveraging LTL, we can impose constraints (like "never enter unsafe zone a") and complex goals ("eventually reach b after collecting c") in a precise, unambiguous manner. This motivates the use of logical specifications to guide the learning process, ensuring safety is maintained by design rather than by trial-and-error reward tuning.

Another limitation of standard RL is the narrow task specificity of learned policies [11, 40]. If an agent learns to "pick up object d and take it to location e", it won't automatically solve "pick up object f and take it to g", despite the similarity. There is a growing realization that *compositionality* is key to generalization in RL [5, 19, 20, 30, 35, 38]. Humans solve novel tasks by composing familiar skills, e.g., a delivery robot that can independently "navigate to a" and "pick up b" should handle "go to a, pick up b" without retraining. In RL research, this insight has led to methods for task composition, where simpler policies or value functions are combined to tackle more complex objectives. Two forms of composition are particularly useful: (1) *Boolean composition* [30], where tasks are combined with the logical operators AND, OR, NOT – for instance, accomplishing task a and task b concurrently – and (2) *Temporal composition* [19, 38], where tasks are executed in sequence (first a then b). Incorporating these forms of composition enables an agent to generalize across a wide range of task specifications defined over a common set of skills.

Contributions. In this paper, we introduce Safe-Comp, a unified framework for safe and generalizable reinforcement learning via policy composition. Our approach leverages formal logic to enforce safety constraints and uses hierarchical policy composition to generalize across tasks. The main contributions of our work are summarized as follows:

Safe Hierarchical RL Framework: We propose a two-level RL framework that separates *skill learning* from *task composition*. At the lower level, the agent learns a set of atomic skills (policies for basic

Proc. of the Adaptive and Learning Agents Workshop (ALA 2026), Aydeniz, Delgrange, Mohammedalamen, Yang (eds.), May 25 – 26, 2026, Paphos, Cyprus, https://alaworkshop2026.github.io/. 2026.

tasks like reaching a region or picking up an object) with safety-aware training. At the higher level, a logical planner composes these skills to satisfy complex task specifications expressed in LTL.

Dual Policy Composition: To pursue safety with best effort, without sacrificing performance, we introduce a dual-policy mechanism. For each skill, the agent learns (i) a *reward-maximizing policy* (optimized for efficiency) and (ii) a *safety-focused policy* (which strictly satisfies all constraints, e.g., by taking detours around hazards). Our framework then composes these policies by dynamically switching to the safe policy only when necessary.

Generalizable Skill Composition: Safe-Comp enables zero-shot generalization to new tasks through Boolean and sequential composition of skills. Given any high-level task defined by an LTL formula (e.g., a combination of sub-goals and safety conditions), we automatically construct a corresponding DFA that captures the task's logical structure. Using this DFA as a guide, our agent performs a graph search (in our case, a Dijkstra-based planning) over skill policies to find a sequence that satisfies the task, with *no further learning required*.

Experiment evaluation: Popular 2D-Navigation simulation experiments and a simple 3D extension are used to evaluate Safe-Comp, where the agents are trained to accomplish a number of tasks containing individual or LTL-combined reach-avoid instructions. The performance is benchmarked on successfully reaching the targets, never touching forbidden regions, and selecting the best sequence of sub-tasks to minimize cost (overall path length or time taken).

The companion code is available at https://github.com/yingyan797/y-compose

2 BACKGROUND

Reinforcement Learning (RL) environments are typically modelled as *Markov Decision Processes* (MDPs). Formally, an MDP is a tuple $M = (S, A, P, R, \gamma)$, where S is a finite set of states; A is a finite set of actions; $P : S \times A \times S \to [0, 1]$ is a deterministic transition function ; $R : S \times A \to \mathbb{R}$ is a reward function; and γ is the discount factor. [1]

An RL agent seeks to maximize the *expected discounted cumulative reward*, $V^\pi(s) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s]$, also called the *value function*, by following some *policy* $\pi : S \to \Delta(A)$, where $\Delta(A)$ denote the set of distributions on actions.

The *Q-function* denotes the expected discounted cumulative reward (under policy π), from state s taking action a,

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s, a_0 = a \right]$$

Optimal policies (a solution to RL) can be derived by iterative applications of the Bellman optimality equation,

$$Q^*(s, a) = \mathbb{E}_{s'} \left[R(s, a) + \gamma \max_{a'} Q^*(s', a') \right]$$

The fixed point of the Bellman optimality equation, $Q^*(s, a)$, denotes the optimal expected discounted cumulative reward from state s, taking action a. Typically, Q-learning algorithms iteratively approximate $Q^*(s, a)$, from experience. Deep Q-Networks (DQN) [29], for example, extend this idea to high-dimensional state spaces. Importantly, because Q-values decompose action utility across states and tasks, they provide a natural foundation for compositional reasoning: primitive Q-functions can be combined to build policies for novel tasks.

[1] Determinism is assumed for clarity and to enable exact graph-based planning in Section 3.4. In stochastic settings, Safe-Comp can be extended with expected-cost edge weights and risk-aware planning; see the discussion under *Application scope*.

3 SAFETY-AWARE COMPOSITIONALITY

We introduce Safe-Comp, a framework for safe and generalizable reinforcement learning that combines *goal-oriented Q-learning*, *dual-policy training*, and *logical task composition* using temporal logic. Safe-Comp enables agents to learn a set of safety-aware skills once, and to generalize compositionally to an extensive class of tasks specified in a temporal logic language we call Reach-Avoid-LTL (RALTL). First we detail the scope of our proposed framework.

Application scope. RL allows for a wide range of applications, depending on how the environment is modelled, our approach, Safe-Comp, is applicable when the following conditions hold:

- Agents can be trained in a simulator – thus enabling random exploration without actually causing any practical danger.
- Generalization to new tasks requires the environment to be static, i.e., it remains unchanged from training to deployment.
- The transition dynamics are deterministic. The optimal sequence of actions of achieving a specific sub-goal must be fixed and used by Dijkstra's algorithm during path planning, otherwise high-level optimality might not be achievable.
- All predicates $p \in AP$ can be expressed in terms of regions in the environment, regardless of the number of physical or logical dimensions of the environment.

Strengths. These assumptions yield convenient guarantees: exact skill learning per label, closed-form Boolean composition, and sound DFA planning. *Restrictiveness.* Determinism and static labelling can be strong in real-world settings; in practice, one can relax them using expected edge costs, robust margins on regions, or chance constraints, at the expense of exact optimality.

Section overview. We first introduce RALTL (syntax and finite-trace semantics) in Section 3.1, including the *sequential-then* operator T. We then describe learning *atomic tasks* and composing *Boolean goals* in Section 3.2 and Section 3.3. Finally, Section 3.4 explains how LTLf-to-DFA translation and graph search yield high-level plans over composed skills.

3.1 Reach-Avoid Linear-time Temporal Logic

To express tasks formally, we introduce the Reach-Avoid-LTL (RALTL) language, a fragment of LTL, interpreted on finite traces, tailored for reinforcement learning.

Syntax. A well-formed RALTL formula over a finite set AP of atomic propositions can be constructed with the following grammar:

$$\begin{aligned} r &::= \top \mid p \mid \neg r \mid r \wedge r \\ \phi &::= rUr \mid \neg\phi \mid \phi \wedge \phi \mid X\phi \mid \phi U\phi \mid \phi T\phi \end{aligned}$$

where $p \in AP$ is an *atomic predicate* (a labelled region, e.g., "goal", "obstacle"); r is a boolean combination of atoms. An *atomic task* rUr'

specifies reach-avoid behaviour: eventually reach region r' while staying in region r ("condition" r and "goal" r'). An RALTL formula ϕ is built from atomic tasks using Boolean and temporal operators: conjunction ($\wedge$), negation ($\neg$), sequencing (T), next (X) and until (U). In addition, the common temporal operators "eventually" (F) and "always" (G), can be derived in the usual way: $F\varphi \equiv \top U \varphi$ and $G \equiv \neg F \neg \varphi$ respectively.

Semantics. In this paper, we consider finite trace semantics for RALTL [10]. Let $\tau = s_0 s_1 \dots s_n$ be a finite word over 2^{AP} and let $(\tau, i) \models \varphi$ denote satisfaction at position $i \geq 0$, and $|\tau| < \infty$ denote the length of the trace.

We assume a labelling function $L : S \to 2^{AP}$ that intuitively assigns to every state $s \in S$, set $L(s) \subseteq AP$ of atoms true in s. The semantics for RALTL is then defined as follows:

$(\tau, i) \models \top$
$(\tau, i) \models p$ iff $p \in L(\tau(i))$
$(\tau, i) \models \neg r$ iff $(\tau, i) \not\models r$
$(\tau, i) \models r \wedge r'$ iff $(\tau, i) \models r$ and $(\tau, i) \models r'$
$(\tau, i) \models r U r'$ iff there exists $j \geq i$ such that $(\tau, i) \models r'$ and for all $k \in [i, j)$: $(\tau, k) \models r$
$(\tau, i) \models \neg\phi$ iff $(\tau, i) \not\models \phi$
$(\tau, i) \models \phi \wedge \phi'$ iff $(\tau, i) \models \phi$ and $(\tau, i) \models \phi'$
$(\tau, i) \models X\,\phi$ iff $|\tau| > i$ and $(\tau, i{+}1) \models \phi$
$(\tau, i) \models \phi\, U\, \phi'$ iff there exists $j \geq i$ such that $(\tau, i) \models \phi'$ and for all $k \in [i, j)$: $(\tau, k) \models \phi$
$(\tau, i) \models \phi\, T\, \phi'$ iff there exists $j \geq i$ such that $(\tau, j) \models \phi'$ and there exists $k \in [i, j)$: $(\tau, k) \models \phi$

RALTL is a *task-oriented fragment* of LTLf. In particular, RALTL restricts the syntax of LTL, where atoms are region formulas and atomic tasks have the structured form rUr', directly matching *reach-avoid* skills learned by the agent. Still, RALTL is suitable to express a wealth of temporal properties, like reachability and (bounded) safety.

Furthermore, since RALTL is a fragment of LTLf, satisfaction is over *finite* traces and formulas compile to *deterministic finite automata* (DFA) with *accepting terminal states* [21]. In contrast, standard LTL over *infinite* traces typically compiles to (non)deterministic Büchi automata with *liveness* acceptance (infinitely often). This finite-trace/DFA setting aligns well with episodic RL: the DFA acts as a runtime monitor of task progress, with each trajectory prefix corresponding to an automaton state, and acceptance corresponding to completed tasks.

Example. Let $AP = \{O, S_1, S_2, S_3\}$ denote an obstacle region O and three goal regions. Consider

$$\varphi \;=\; G\,\neg O \;\wedge\; (((FS_1) \;\vee\; (FS_2))\; T\; (FS_3))$$

Intuitively: *always avoid* O; *then* reach either S_1 or S_2 at least once before *eventually* reaching S_3. Under the finite-trace semantics above, $(\tau, i) \models \varphi$ iff every position up to acceptance satisfies $\neg O$, there exists a j with $(\tau, j) \models (FS_3)$, and there exists some $k < j$ with $(\tau, k) \models (FS_1) \vee (FS_2)$. When compiled to a DFA, the disjunction is placed in DNF and becomes *two* alternative outgoing edges (choose S_1-first or S_2-first), after which the automaton requires completion of (FS_3); any visit to O leads to a rejecting sink. The edge policies for (FS_1), (FS_2), and (FS_3) are obtained from the learned atomic reachers, while the choice between S_1 or S_2 is resolved by the planner using their composed Q-values (Section 3.3) and the DFA search (Section 3.4).

3.2 Algorithms for Safety-aware Policies

Algorithm 1 Safe Q-Learning with Policy Composition Overview

Phase 1 & 2: Direct and Safe Q-value functions - $Q(s, a, g)$
for each iteration **do**
  for each sub-goal g **do**
    repeat
      Select optimal action from $Q(s, a, g)$ or $Q_{safe}(s, a, g)$.
      Compute sub-goal g reward or penalty.
      Update $Q(s, a, g)$ or $Q_{safe}(s, a, g)$.
    until reaching g

Phase 3: Policy Composition - $Q(s, a, g) \to Q(s, a)$
Disjunction: $Q(s, a, \vee_{i\in[1,n]} g_i) = max_{i\in[1,n]} Q(s, a, g_i)$
Conjunction: $Q(s, a, \wedge_{i\in[1,n]} g_i) = \sum_{i\in[1,n]} Q(s, a, g_i)$
Negation: $Q(s, a, \neg g_i) = \max_{j\neq i \in[1,n]} Q(s, a, g_j)$
Replacement: $Q_{direct}(s, a) \leftarrow Q_{safe}(s, a)$ for s in goal-partition

Phase 4: Safe Policy Replacement (shielding)
Compute composed $Q_{direct}(s, a)$ and $Q_{safe}(s, a)$
Record processed
Initialize $Q_{shield}(s, a) \leftarrow Q_{direct}(s, a)$
for each unprocessed state s **do**
  Initialize empty visited states set
  repeat
    Select and take optimal action according to Q_{shield}
    if reaching a state that is: (1) out of the safety condition region **OR** (2) visited but not processed (to prevent loops) **then**
      Replace Q_{shield} with Q_{safe} and re-select action
  until reaching the goal or any processed state
  Add the whole trajectory to processed states

To learn policies for getting to environment-labelled regions, we train extended Q-functions $Q(s, a, g)$. This is split into two phases: Phase 1 learns *direct/efficient* reach-avoid primitives; Phase 2 learns *safe/constraint-satisfying* reach-avoid primitives. This setup explicitly considers both optimality and safety as critical components. Once the primitives have been learnt we can generalize to Boolean combinations *without additional training* (zero-shot) (c.f., Section 3.3) and temporal (sequential) combinations (c.f., Section 3.4).

Inspired by [30], we use *goal-oriented Q-learning* which uses extended Q-functions $Q(s, a, g)$ to represent policies. In Safe-Comp we train two complementary phases (direct and safe), summarized as follows:

Direct Policy (Phase 1). We randomly initialize the agent at different starting states and iteratively update the Q-function $Q(s, a, g)$ encoding the value of reaching sub-goal g from state s by action a. A positive reward is given after reaching a terminal state p labelled with the current sub-goal g (i.e., $p \in L(g)$). No rewards or penalties for any other states. Consequently, the arg max policy derived from $Q(s, a, g)$ takes the agent directly (without considering avoidance of other regions) from state s to goal state g.

Safe Policy (Phase 2). In parallel the agent will learn an additional extended Q-function $Q_{safe}(s, a, g)$. Again we randomly initialize the agent, but the goal is now to reach label g while either staying in or avoiding all other labelled regions, depending on the starting state. Again, a positive reward is provided upon reaching label g, and a penalty for breaking the above constraint. This dual-policy design ensures that every sub-goal has both an optimal and a safety-guaranteed policy, if individual training stage found existing.

3.3 Boolean Task Composition

For the Boolean composition of tasks and policies, we assume the same environment structure with a fixed state space, with tasks only differing in their goal conditions [30, 35]. Suppose an agent learns Q-functions for a set of *primitive tasks* (e.g., "reach goal A", "reach goal B"). Often these functions can be composed to form new tasks without retraining. For example, to achieve "A or B" (*disjunction*), one can take the maximum value across primitives: $Q_{A\vee B} = \max(Q_A(s, a), Q_B(s, a))$.

For more complex compositions, e.g., *conjunction*, one could take an average over the constituent primitives. However, this may result in a suboptimal composition [13] if the two objectives do not have a significant overlap in Q-function space, a gap that can in fact be quantified [16]. Rather, [30] proposed the extended Q-functions $Q : S\times G\times A \to \mathbb{R}$, where $g \in G$ is a goal state, and $G \subseteq S$ is the goal space. $Q(s, g, a)$ encodes the value of taking action a in state s with the intention to reach g. If the agent terminates at a goal belonging to a different primitive, it receives the worst possible penalty $r_{\min}$. This ensures the agent learns values for all goals simultaneously rather than only the closest one.

For task composition in Safe-Comp, similarly, we use extended Q-functions $Q(s, a, g)$ (indexed by sub-goals g) reducing them into a regular Q-function $Q(s, a)$ for a Boolean combination of sub-goals. The Boolean expressions are normalized into *disjunctive normal form* (DNF); thus, the semantics of such *Boolean expressions* are Disjunctions of Conjunctions of labelled regions (or their negation), where Negation is only applied to individual sub-goals.

Conjunction (AND):. If the task requires reaching $g_1 \wedge g_2$, then the associated region is the *intersection* of the two sub-goal regions. In Q-function space, we aggregate Q-values by summation over the sub-goal dimension:

$$Q(s, a, g_1 \wedge g_2 \wedge ...g_n) = \sum_{g\in[g_1,g_2,...,g_n]} Q(s, a, g),$$

This captures the need to satisfy both/all sub-goals eventually.

Disjunction (OR):. For $g_1 \vee g_2$, the associated region is the *union* of sub-goals. The composed Q-value is obtained by maximization over the corresponding sub-goal indices:

$$Q(s, a, g_1 \vee g_2 \vee ...g_n) = \max_{g\in[g_1,g_2,...,g_n]} Q(s, a, g)$$

This ensures that whichever goal offers the best outcome is prioritized, yielding an efficient disjunctive behaviour.

Negation (NOT):. Negation, i.e., $\neg g$ corresponds to the complement of a sub-goal: i.e., all other sub-goals. In Q-space, this is derived from baseline tasks by taking the maximum of the complement set:

$$Q(s, a, \neg g_i) = \max_{j\in[1,2,..,n],j\neq i} Q(s, a, g_j)$$

Importantly, DNF normalization ensures nested negations are eliminated, bypassing the unsupported induction step due to the limitation of the goal-oriented design in finding negated composed goals. For conjunction and disjunction, conversely, the induction step can be performed normally on the resulting Q functions.

Replacement. After obtaining a reduced policy for both direct and safe Q-values, an additional step is required: for states within all sub-goals connected to the composed region (goal-partition), values from the direct Q-function are overwritten with those from the safe Q-function, ensuring safety consistency.

Through this process, Boolean task composition systematically reduces label-based formulas to standard Q-functions executable as policies (via arg max). Algorithm 1 (Lines 10–15) summarizes the construction for disjunction, conjunction, negation, and the replacement step.

Safe Policy Replacement/Shielding (Phase 4). While Boolean composition yields correct logical behaviour, policies composed solely from direct Q-values may violate safety constraints, and those based only on safe Q-values are often overly conservative. To balance these objectives, Safe-Comp introduces a trajectory-based shielding mechanism (Algorithm 1, Lines 16–27).

The combined policy is first set to Q_{direct}. At each step following the starting state, we evaluate whether the next action under Q_{direct} would lead to a violation (by querying the environment to see if the reaching state is outside the safety condition). Upon detecting a violation, the action is rejected and control switches to Q_{safe} to adhere to learned safety constraints. Once the trajectory re-enters a safe regime, control restores to Q_{direct}. To avoid infinite loops or repeated checks, the algorithm maintains sets of processed and visited states, ensuring convergence of the replacement procedure. This mechanism ensures that the combined policy is safe whenever Q_{safe} itself is safe and valid (free of dead ends or loops). In practice, we expect the resulting trajectories achieve higher efficiency than Q_{safe}, since they retain Q_{direct} whenever no risk is present.

3.4 Task Combination with LTLf and DFA

Once policies are obtained for individual Atomic Tasks, Safe-Comp constructs a *Deterministic Finite Automaton (DFA)* to capture complex tasks specified in Linear Temporal Logic on Finite Traces (LTLf/RALTL). The DFA provides a structured execution plan, where each automaton state represents task progress, and transitions are labelled by Boolean combinations of atomic tasks. The overall policy for the higher-level task is then derived by planning over this DFA.

Formula to DFA via MONA Representation. The translation from an LTLf formula φ into a DFA $\mathcal{A}_\varphi$ follows the approach of Fuggitti [12], with extensions to handle the *Then* operator and bounded temporal constructs. The first step is conversion into the *MONA* representation [41], a logical encoding suitable for automata construction. MONA expresses each sub-formula recursively: new propositional variables are introduced for sub-goals, temporal bounds are explicitly declared, and truth values are propagated stepwise. The

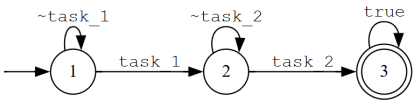

Figure 1: DFA for the sequential combination of two tasks.

result is a machine-readable format that preserves the semantics of φ over finite traces.

Transition Matrix in DNF.. From the MONA representation, a ternary truth table is generated, where entries are $\{1, 0, X\}$ denoting True, False, or Unrelated, respectively. Rows correspond to possible state transitions, while columns capture the truth values of atomic tasks. These truth values are grouped into Boolean expressions in *disjunctive normal form (DNF)*, ensuring that each transition condition is expressed as a disjunction of conjunctions of literals. The DNF representation is crucial: it guarantees that every non-deterministic choice in the task specification (disjunction) is made explicit as separate edges in the DFA.

Example. For the formula $task_1\ T\ task_2$, the transition matrix is given in Table 1. This construction yields three DFA states: the initial state, an intermediate progress state, and an accepting state. The transition conditions in DNF clarify exactly which task completions advance the automaton.

Table 1: The transition matrix for $\varphi = r_1\ T\ r_2$

Transition	r_1	r_2
1⟶1	0	X
1⟶2	1	X
2⟶2	X	0
2⟶3	X	1
3⟶3	X	X

State	1	2	3
1	$\neg r_1$	r_1	-
2	-	$\neg r_2$	r_2
3	-	-	$\top$

The matrix is then used to to create the DFA, which is illustrated in Figure 1.

Policies for DFA Edges. As discussed in the DFA task construction, which uses DNF for transition conditions, each edge is now labelled by a Conjunction of original or negated atomic tasks (α). To execute an edge, Safe-Comp computes a policy for that DNF formula:

- **Negated Atomic Tasks:** If $\alpha = cUg$, then enforcing avoidance of g, i.e., $G(\neg g)$ is a sufficient condition for the negated atomic task $\neg\alpha$.
- **Conjunctive Atomic Tasks:** For $(c_1 U g_1) \wedge (c_2 U g_2)$, the agent must remain in $c_1 \cap c_2$ and reach g_1 and g_2 in any order. This is decomposed into a sequence of atomic subtasks:

$$\alpha_1 := (c_1 \wedge c_2) U g_1, \quad \alpha_2 := (c_1 \wedge c_2) U G_2.$$

The DFA edge policy is then obtained by computing the shortest permutation of subtask completions. This reduces to a variant of the travelling salesman problem (TSP), solvable efficiently by dynamic programming with bit-masking [23]. The result ensures correctness while controlling complexity.

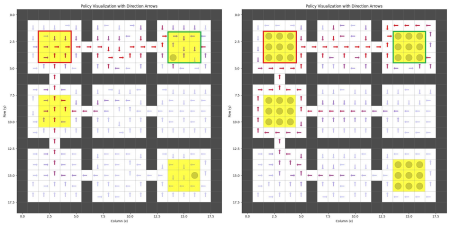

Figure 2: A comparison of direct (L) and safe (R) policy for the Atomic Task ($\neg S_1\ Until\ S_3$).)

High-Level Path Planning. Once edge policies are defined, the DFA is transformed into a weighted graph where nodes are pairs (q, s) of DFA state q and environment state s. Edge costs are given by the expected path length (or value) of the composed Q-function for the edge's formula. The global policy is then computed via Dijkstra's algorithm:

$$\pi_\varphi = \arg \min_{\tau \in Paths(\mathcal{A}_\varphi)} \sum_{e \in \tau} cost(Q_e).$$

This yields a safe and efficient execution trace that respects the task formula φ. Notably, edge policies are computed *on demand*, reducing computational overhead for large DFAs, similar to lazy evaluation in SPECTRL [20].

Summary. Safe-Comp leverages LTLf-to-DFA translation to reduce high-level task specifications into structured sequences of Boolean and atomic sub-goals. By combining safe Q-learning with edge policies for Boolean formulas and global DFA search, the agent can solve arbitrarily complex tasks *compositionally*, while following safety constraints.

4 EXPERIMENTAL EVALUATION

In this section we evaluate the Safe-Comp approach experimentally. To do so, we consider discrete 2D tabular reach-avoid environments, as well as an extension to a 3D environments. As evaluation metrics we analyse policy validity, safety compliance, optimality (trajectory length), and generalizability (number of tasks achievable with upfront training).

4.1 Evaluating Atomic Tasks

A popular reach-avoid environment used by many research is 9-Rooms, as shown in Figure 12, 2 and 3, where some rooms connected to the adjacent ones via a narrow passage. Atomic tasks involve travelling from the bottom left room to some marked sub-goals while avoiding others, with Moving Up, Down, Left, and Right as allowed actions.

The Atomic Task being tested is "Reach S_3 (top right) ensuring S_1 (top left) is avoided: $\neg S_1\ US3$". In Figure 2 and 3, a color-arrow Q-value matrix is visualized for direct, safe, and combined policies. Arrow direction indicates the best action at each state, and the color represents the relative magnitude of Q values (red higher, blue lower).

Figure 2 shows the Direct policy takes the shortest path towards the goal (top right), but may violate the safety constraint that required S_1 (top left) to be avoided. On the other hand, Safe policy

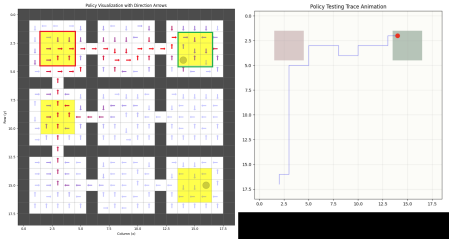

Figure 3: Combined Direct and Safe policy for the Atomic Task $\neg S1\ Until\ S3$ and a trajectory of policy rollout.

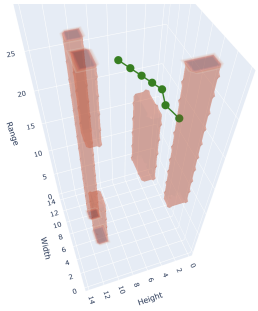

Figure 4: 3D reach-avoid experiment trajectory: Reach the right sub-goal while avoiding the middle one, both spanning a range on the 3rd dimension

avoids all other sub-goals until S_3 reached - safe and valid, but result in the agent taking an unnecessary detour around O (left room), hence less optimal.

Next, safety policy replacement (shielding) mitigates both extremes and results in a much more reasonable policy. Figure 3 shows the combined policy, the unnecessary detour around O is corrected, and the safety requirement is still satisfied. Starting from coordinate (17,2), the ground truth shortest solution takes 25 steps, and the combined policy takes 28 steps. In comparison, the safe policy would need 30 steps.

4.2 Atomic Task in 3D

An additional dimension is added to the environment, which can represent not just physical space, but also logical concepts like resource availability (battery or fuel), load (number of passengers), etc.

Fig. 4 shows a trajectory computer for a simple reach-avoid atomic task, where the trajectory successfully avoids the middle region (passing the region above the forbidden range) until reaching the goal in the accepted range.

4.3 Evaluating Boolean Composition

Another experiment is to test Boolean composition in a custom environment with 3 overlapping sub-goals. Fig 5). The first figure for conjunction, and the second for disjunction. As shown, policy arrows for the conjunction task eventually lead to the "AND" composition of 2 sub-goal regions, and those for the disjunction task lead to the whichever of two sub-goal is closer in general.

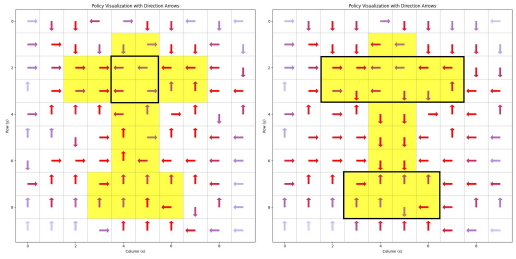

Figure 5: Policy for goal conjunction (L) and disjunction (R)

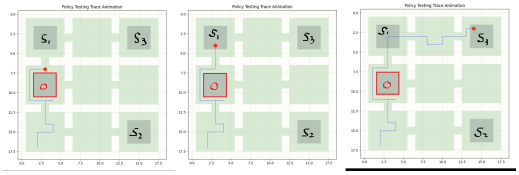

Figure 6: Trajectory of 9-room DFA complex task planning

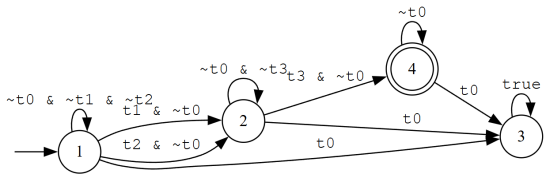

Figure 7: DFA graph of formula $G(\neg t0) \wedge ((t1 \vee t2)\ T\ t3)$

4.4 Evaluating DFA and Path Finding

The following set of experiments is for LTLf task construction and DFA optimal path finding, where task specifications can include complex logic like global safety conditions, sequential composition, non-deterministic choice (complete any accepting sub-task), and conjunction (complete multiple sub-tasks in any order).

Experiment 1 - 9-room. The first test requires the agent to perform the following sub-tasks in order (same as in SPECTRL [19]):

Step	Sub-Task	Logic
1	Start from bottom left room	–
2	Always avoid region O	global restriction
3	Reach either S_1 or S_2	non-deterministic choice
4	Then reach S_3	sequential composition

SPECTRL formula expresses the task as:

$$achieve((reach(S_1)\ or\ reach(S_2));\ reach(S_3)\ ensuring\ (avoid(O))$$

Equivalently in RALTL, the formula is

$$G(\neg t0) \wedge ((t1 \vee t2)\ T\ t3)$$

where 4 Atomic Tasks are constructed: ($t_0 := F(O)$, $t_1 := F(S_1)$, $t_2 := F(S_2)$, $t_3 := F(S_3)$).

Fig. 7 shows the DFA drawn from the formula, indicating that globally dangerous paths (t_0, reaching O) always rejected, and both viable Atomic Tasks (t_1 - via S_1 and t_2 - via S_2) supported as non-deterministic choices, though one shorter than another.

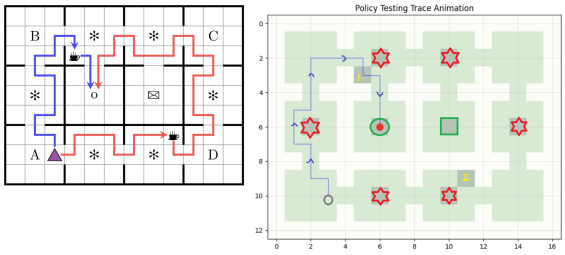

Figure 8: Office grid 2D navigation task 1 - picking up coffee, mail, and delivering to the office without hitting obstacles. Left: blue path is optimal. Right: Actual result

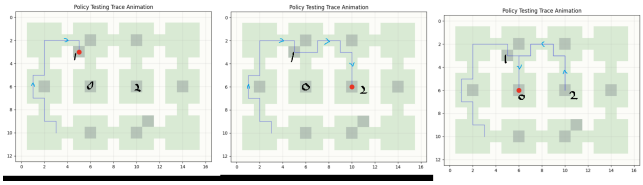

Figure 9: Office grid 2D navigation task 2 - Delivering a coffee and a mail to the office (in any order), avoiding obstacles

Next, Dijkstra shortest path planning successfully selected t_1, the shorter path, instead of t_2, as shown in the trajectories, leading to a significantly less number of steps (37 steps; Otherwise, the agent will waste at least another 20 steps in travelling to S_2 and returning to the starting room).

Experiment 2 - office grid. , taken from [18], as shown in Fig.8). The grid contains 12 rooms, some connected to another via a passage. There are coffee, mail, obstacles, and office (O) labels in this environment. Multiple tasks can be solved by the agent. Task 1: deliver one of the 2 coffee to the office, without hitting into obstacles ('*' shape). All Atomic Tasks needed for the DFA are – t0: $F(O)$; t1: $F(coffee_1)$; t2: $F(coffee_2)$; t3: $F(mail)$; t4: $F(obstacle)$; The tasks specification formula is then defined as:

$$G(\neg t_4) \wedge ((t_1 \vee t_2)\ T\ t_0) \tag{1}$$

This task, similarly, involves a non-deterministic choice for moving to either coffee locations. Result in Fig. 8 shows that the agent successfully determines the shorter of the 2 valid paths to complete the task while avoiding all obstacles.

Task 2: deliver a coffee and a mail to the office in any order. Atomic Tasks remain the same, but the complete task specification can be written as follows:

$$G(\neg t_4) \wedge ((t_1 \vee t_2)\ T\ t_0) \wedge (t_3\ T\ t_0) \tag{2}$$

The formula clearly involves task conjunction, allowing "coffee to office" and "mail to office" completed in either order, or simultaneously. The resulting trajectory (for each step) is shown in 9, which indicates the optimal task order is determined successfully.

Experiment 3 - Minecraft task generalization. This test is to further demonstrate the range of tasks achievable with our RALTL

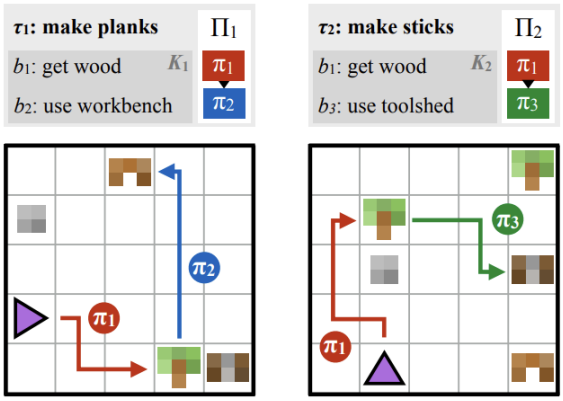

Figure 10: Example of Minecraft grid task composition experiments, used in [18] as a test

#	Task	Temporal Logic Formula
1	make plank	Gw T Uts
2	make stick	Gw T Uwb
3	make cloth	Gg T Uf
4	make rope	Gg T Uts
5	make bridge	(Gi T Uf) & (Gw T Uf)
6	make bed	(Gw T Uts) & (Gg T Uwb)
7	make axe	(Gw T Uwb) & (Gi T Uts)
8	make shears	(Gw T Uwb) & (Gi T Uwb)
9	get gold	((Gi T Uf) & (Gw T Uf)) T Ub
10	get gem	((Gw T Uwb) & (Gi T Uts)) T Ua

specifications. Task and environment settings are defined in a discrete 2D grid, analogous to office grid, where components like "Get wood/Gw", "Get iron/Gi", "Get grass/Gg", "Use toolshed/Uts", "Use workbench/Uwb", and "Use bridge/Ub", etc., are Atomic Task for reaching a labelled position. Resulting DFA and trajectories are also very similar to office grid settings.

4.5 Discussion of Limitations

1. Safety shielding greedy optimality. Traversing the environment with the purpose of combining Direct and Safe policy models, the agent switches to safe behaviour only at the immediate step of predicted violation. This greedy approach may possibly result in suboptimal detours, and more optimal solution could be obtained by formulating the replacement step as a modified Bellman update, anticipating violations earlier in the trajectory and incorporating them into value propagation. This remains a promising direction for future research.

2. Reach and avoid interfering regions - safety requirement. During the initial dual policy learning stage, Safe policy is learned by keeping the agent exploring within area with no labels (external) or area within one or more connected sub-goals (internal), analogous to sea and islands respectively, where a particular sub-goal can be an "island" itself or just part of the island. Therefore, the agent only learns to find trajectories within the same "island" or from "sea" directly to the sub-goal. When an atomic task requires avoiding

one part of the internal (a Boolean combination of sub-goals) until reaching the other part, the "avoid" constraint is not learnt as they belong to the same "island". In conclusion, the practitioners are supposed to design the label assignments in a way that doesn't hit this limitation, for example as in the 9-rooms, Office grid, or Minecraft experiments where "reach" and "avoid" regions are always disjoint from each other, separated with externals, no matter what Boolean composition is involved.

5 RELATED WORK

In this section we discuss the works most related to the present contribution in safe reinforcement learning, RL with logical constraints, and task policies composition.

Safe Reinforcement Learning. The Constrained Markov Decision Process (CMDP) framework introduces constraints on agent behaviour (modelled as cost functions) alongside the reward [1, 3, 8, 26, 31, 34, 39]. Constrained Policy Optimization (CPO) [1] is a policy search algorithm that provides guarantees of near-constraint satisfaction at each iteration. Subsequent works have built on this idea using Lagrange multiplier techniques and primal-dual updates to balance reward and constraint satisfaction [31]. A key limitation of the CMDP framework is that it only enforces (cumulative) constraints in expectation. In contrast, *shielding* and *runtime verification* have emerged as practical solutions to enforce absolute (probability 1) safety during policy execution [2, 6]. Shielding involves an external module that monitors the agent's chosen action and overrides or "filters" any action that would lead to a forbidden state. Classical approaches to shielding work by synthesizing shields from temporal specifications (often LTL-safety specifications) [2, 6]. These methods ensure safety *by construction*, and with minimal impedance on the policy's actions, although they demand a hand-crafted safety abstraction of the environment dynamics, often limiting shielding to small scale and simple scenarios.

RL with Logical Constraints. A substantial body of work integrates logical specifications into RL to manage complex objectives and constraints [9, 15, 25, 32, 36, 37]. One line of research uses formal languages (such as LTL) to shape reward functions or restructure the learning process [9, 15, 25]. For example, *Logically-Constrained RL* [15] converts an LTL specification into an automaton and augments the reward signal based on the automaton's state, guiding the agent toward satisfying the formula by rewarding progress toward accepting states. Similarly, [25] translate LTL or signal temporal logic directly into rewards, so that fulfilling the specification yields high reward, while violations incur penalty.

A closely related idea is the use of *reward machines* [17, 18]; extended Mealy machines that encode the reward logic and track the agent's progress through sub-tasks. By exposing the internal structure of the reward (e.g., which sub-goal is currently needed), reward machines allow the agent to learn more efficiently and handle non-Markovian rewards that depend on event sequences. More generally, [7] explores formal languages for reward specification, showing that temporal logics and regular expressions can be compiled into automata that structure RL objectives.

These approaches all exploit logical structure for better reward design and credit assignment, but they typically remain limited to shaping individual task rewards, rather than addressing large-scale compositionality across tasks.

Compositional Task Policies. Policy composition seeks to generalize across tasks by reusing learned behaviours. On key aspect is *Boolean task composition*, where tasks are combined using logical operators. For example, an agent that learned policies for task a and b can solve the conjunction $a \wedge b$ by executing both policies appropriately [30]. Earlier work focused on linear composition of value functions, where a weighted sum of known Q-functions yields a new policy that balances objectives [35].

Another important aspect is sequential (temporal) composition of tasks. In hierarchical RL, the notion of options [33] allows chaining skills in sequence (e.g., complete option a, then option b). Building on this idea, recent neuro-symbolic RL approaches use logic to guide skill sequencing [5, 19, 20, 22, 38]. SPECTRL [19] introduced a high-level specification language that compiles logical task formulas into a hierarchy of sub-policies. Its extension [20] supports compositional specifications, combining multiple formulas that are satisfied by orchestrating different modules. Similarly, [22] proposed a latent-goal agent using a simplified temporal logic (SATTL) to activate pre-trained sub-policies based on current logical sub-goals. More recent work explores zero-shot generalization for unseen logical tasks: for instance, [38] use implicit planning over previously learned option policies, while Comp-LTL [5] constructs a symbolic transition system and plans over pre-trained primitives to satisfy new LTL specifications.

Our approach belongs to this compositional paradigm. We explicitly support both Boolean and sequential composition of skills via LTL specifications: a conjunction of goals, a disjunction (choice), or an ordered sequence can all be encoded in LTL and executed by composing the corresponding primitives. Like Comp-LTL [5], our framework achieves zero-shot policy composition for unseen tasks, but we go further by integrating a safety layer (via dual policies) and validating its effectiveness in safety-critical environments, where naive composition may otherwise produce hazardous behaviours.

6 CONCLUSIONS

In this paper we introduced Safe-Comp, a unified framework for safe and generalizable reinforcement learning via policy composition. We first introduced Reach-Avoid-LTL (RALTL) as a rich task specification language that builds on atomic reach-avoid goals by using the full power of Linear-time Temporal Logics. Based on RALTL, we proposed a two-level RL framework, where agents learns a set of atomic skills at the lower level, while at the higher level, a logical planner composes these skills to satisfy complex tasks expressed in RALTL. In particular, to follow safety rules without sacrificing performance, for each skill, the agent learns a reward-maximizing policy and a safety-focused policy. Then, Safe-Comp enables zero-shot generalization by leveraging on DFAs that captures the task's logical structure. Finally, we evaluated our approach experimentally, on environments taken from the literature on task composition and reward machine, showing significant preliminary results.

Acknowledgements. The research described in this paper was partially supported by the EPSRC (grant number EP/X015823/1).

REFERENCES

[1] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. 2017. Constrained policy optimization. In *International conference on machine learning*. PMLR, 22–31.

[2] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. 2018. Safe reinforcement learning via shielding. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[3] Eitan Altman. 2021. *Constrained Markov decision processes*. Routledge.

[4] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. 2016. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565* (2016).

[5] Taylor Bergeron, Zachary Serlin, and Kevin Leahy. 2024. Comp-LTL: Temporal Logic Planning via Zero-Shot Policy Composition. *arXiv preprint arXiv:2408.04215* (2024).

[6] Roderick Bloem, Bettina Könighofer, Robert Könighofer, and Chao Wang. 2015. Shield synthesis: Runtime enforcement for reactive systems. In *International conference on tools and algorithms for the construction and analysis of systems*. Springer, 533–548.

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

[8] Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. 2018. Risk-constrained reinforcement learning with percentile risk criteria. *Journal of Machine Learning Research* 18, 167 (2018), 1–51.

[9] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. 2020. Restraining bolts for reinforcement learning agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 13659–13662.

[10] Giuseppe De Giacomo, Moshe Y Vardi, et al. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces.. In *Ijcai*, Vol. 13. 854–860.

[11] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. 2018. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070* (2018).

[12] Francesco Fuggitti. 2019. *LTLf2DFA*. https://doi.org/10.5281/zenodo.3888410

[13] Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. 2018. Composable deep reinforcement learning for robotic manipulation. In *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 6244–6251.

[14] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. 2017. Reinforcement learning with deep energy-based policies. In *International conference on machine learning*. PMLR, 1352–1361.

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

[16] Jonathan Hunt, Andre Barreto, Timothy Lillicrap, and Nicolas Heess. 2019. Composing Entropic Policies using Divergence Correction. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 2911–2920. https://proceedings.mlr.press/v97/hunt19a.html

[17] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. 2018. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2107–2116.

[18] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. 2022. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research* 73 (2022), 173–208.

[19] Kishor Jothimurugan, Rajeev Alur, and Osbert Bastani. 2019. A composable specification language for reinforcement learning tasks. *Advances in Neural Information Processing Systems* 32 (2019).

[20] Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. 2021. Compositional reinforcement learning from logical specifications. *Advances in Neural Information Processing Systems* 34 (2021), 10026–10039.

[21] Nils Klarlund, Anders Mller, and Nils Mller. 2001. MONA Version 1.4 - User Manual. (02 2001).

[22] Borja G León, Murray Shanahan, and Francesco Belardinelli. 2021. In a nutshell, the human asked for this: Latent goals for following temporal specifications. *arXiv preprint arXiv:2110.09461* (2021).

[23] Gonzalo Lera-Romero, Juan José Miranda-Bront, and Francisco J. Soulignac. 2014. Dynamic programming for the time-dependent traveling salesman problem with time windows. – (2014).

[24] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. 2016. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research* 17, 39 (2016), 1–40.

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

[26] Qingkai Liang, Fanyu Que, and Eytan Modiano. 2018. Accelerated primal-dual policy optimization for safe reinforcement learning. *arXiv preprint arXiv:1802.06480* (2018).

[27] Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew J. Hausknecht, and Michael Bowling. 2018. Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents. *Journal of Artificial Intelligence Research* 61 (2018), 523–562.

[28] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM workshop on hot topics in networks*. 50–56.

[29] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[30] Geraud Nangue Tasse, Steven James, and Benjamin Rosman. 2020. A boolean task algebra for reinforcement learning. *Advances in Neural Information Processing Systems* 33 (2020), 9497–9507.

[31] Alex Ray, Joshua Achiam, and Dario Amodei. 2019. Benchmarking safe exploration in deep reinforcement learning. *arXiv preprint arXiv:1910.01708* 7, 1 (2019), 2.

[32] Ameesh Shah, Cameron Voloshin, Chenxi Yang, Abhinav Verma, Swarat Chaudhuri, and Sanjit A Seshia. 2024. Deep Policy Optimization with Temporal Logic Constraints. *arXiv preprint arXiv:2404.11578* (2024).

[33] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (second ed.). The MIT Press. http://incompleteideas.net/book/the-book-2nd.html

[34] Chen Tessler, Daniel J Mankowitz, and Shie Mannor. 2018. Reward constrained policy optimization. *arXiv preprint arXiv:1805.11074* (2018).

[35] Benjamin Van Niekerk, Steven James, Adam Earle, and Benjamin Rosman. 2019. Composing value functions in reinforcement learning. In *International conference on machine learning*. PMLR, 6401–6409.

[36] Cameron Voloshin, Hoang Le, Swarat Chaudhuri, and Yisong Yue. 2022. Policy optimization with linear temporal logic constraints. *Advances in Neural Information Processing Systems* 35 (2022), 17690–17702.

[37] Cameron Voloshin, Abhinav Verma, and Yisong Yue. 2023. Eventual discounting temporal logic counterfactual experience replay. In *International Conference on Machine Learning*. PMLR, 35137–35150.

[38] Duo Xu and Faramarz Fekri. 2024. Generalization of Compositional Tasks with Logical Specification via Implicit Planning. *arXiv preprint arXiv:2410.09686* (2024).

[39] Tsung-Yen Yang, Justinian Rosca, Karthik Narasimhan, and Peter J Ramadge. 2020. Projection-based constrained policy optimization. *arXiv preprint arXiv:2010.03152* (2020).

[40] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. 2020. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*. PMLR, 1094–1100.

[41] S Zhu, G Pu, and MY Vardi. 2019. First-Order vs. Second-Order Encodings for LTLf-to-Automata Translation. arXiv 2019. *arXiv preprint arXiv:1901.06108* (2019).

A SUPPLEMENTARY MATERIAL

A.1 Detailed algorithms and data structure

1. DFA edge policy (for conjunction tasks) ans Dijkstra path planning. Both policies are derived through evaluation of trajectory costs by simulating the agent from one particular starting state. No training is needed, and a planned path to accomplish a selected sequence of sub-tasks is produced in seconds after the starting state is given (inference only).

Algorithm 2 DFA Edge Conjunction tasks (permutation)

Require: all condition regions C_{list}, all goal regions G_{list}, $starting\_state$

$condition \leftarrow \cap(C \in C_{list})$
Initialize $optimal\_policy \leftarrow \{\}$, $terminal\_state \leftarrow empty$
for each $permutation$ of G_{list} **do**
  Initialize $state \leftarrow starting\_state$, $policy \leftarrow \{\}$
  for each $G \in permutation$ **do**
    $AT \leftarrow (condition\ U\ G)$
    $actions, final\_state \leftarrow FollowAtomicPolicy(AT, condition, G, state)$
    $policy \leftarrow policy \cup actions$
    $state \leftarrow final\_state$
  if $optimal\_policy$ is empty **or** $policy.length < optimal\_policy.length$ **then**
    $optimal\_policy \leftarrow policy$, $terminal\_state \leftarrow state$
return $optimal\_policy$, $terminal\_state$

Algorithm 3 Dijkstra DFA path search

Require: DFA, $init\_location$, $start\_state$ $accepting\_states$

Initialize $visited \leftarrow \{\}$
$queue \leftarrow \{(start\_state, init\_location', 0, \{\})\}$
while $queue$ is not empty **do**
  $node \leftarrow$ remove from $queue$ with minimum tot_cost
  $(state, location, tot\_cost, path) \leftarrow node$
  if $state \in accepting\_states$ **then**
    return $path$
  if $(state, location) \notin visited$ **or** $visited[(state, location)].tot\_cost > tot\_cost$ **then**
    $visited[(state, location)] \leftarrow node$
    for each $(state', edge) \in DFA[state]$ **do**
      if $state' \neq state$ **then**
        $(actions, location') \leftarrow EdgePolicy(edge, location)$
        $cost' \leftarrow tot\_cost + actions.length$
        $path' \leftarrow path \cup \{(state', actions)\}$
        if $(state', location') \notin visited$ **or** $visited[(state', location')].tot\_cost > cost'$ **then**
          $queue \leftarrow queue \cup \{(state', location', cost', path')\}$

2. Sketch of MONA LTL representation examples. This is the encoding method of LTL in a programmatic way, used by computer programs to generate state diagrams and DFA subsequently.

Then logic:

$$(\exists_1 v_1 : v_1 \in \$ \wedge 0 < v_1 \leq max(\$) \wedge mona(v_1, \phi_2) \wedge \exists_1 v_2 : v_2 \in \$ \wedge 0 \leq v_2 < v_1 \wedge mona(v_2, \phi_1));$$

Until logic

$$(\exists_1 v_1 : v_1 \in \$ \wedge 0 < v_1 \leq max(\$) \wedge mona(v_1, \phi_2) \wedge \forall v_2 : v_2 \in \$ \wedge 0 \leq v_2 \leq v_1 \wedge mona(v_2, \phi_1));$$

Next Logic

$$\exists_1 v_1 : v_1 \in \$ \wedge v_1 = 1 \wedge v_1 \leq max(\$) \wedge mona(v_1, \phi)$$

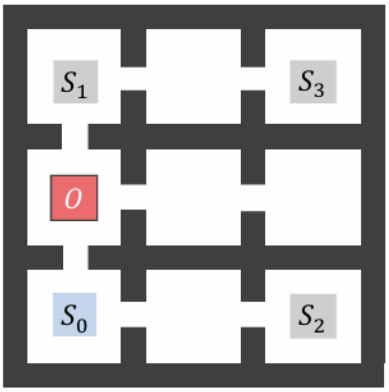

Figure 11: 9-room reach-avoid experiment

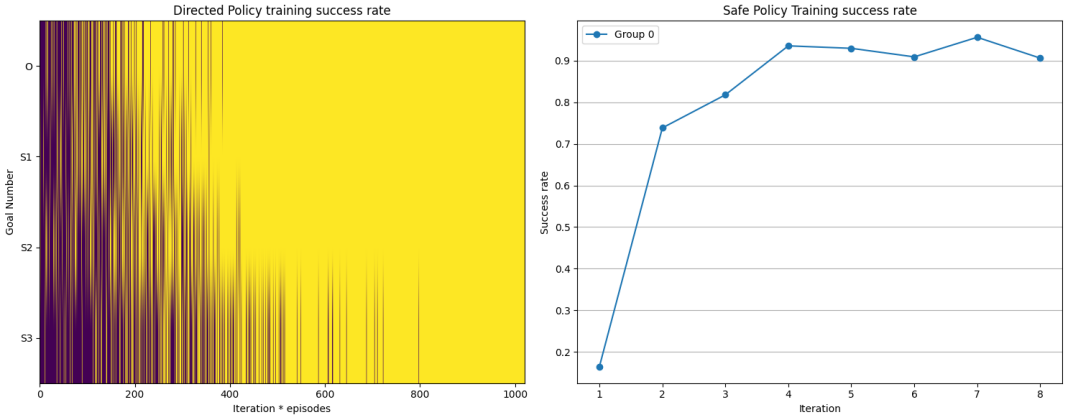

Figure 12: Dual policy training: Phase 1 (Left - Direct policy) records the successful (light) and failed (dark) episodes for each of the 4 sub-goals on y-axis. Phase 2 (Right - Safe-policy) records the success rate starting from any random non-goal state. Completing within 150 steps is considered successful

A.2 Experiment settings

Figure 11 is the original 9-room experiment layout used by popular research. In addition, the training procedure for initial dual-policy with extended Q-Learning (goal-oriented) is recorded as shown in the plots of 12. The left plot is for Direct policy, where the 4 sections of y-axis represent learning to reach each sub-goal, and the x-axis is episode number. Vertical lines of Dark color suggests an episode failing to reach the sub-goal within certain step limit, and light color otherwise. The plot indicates that after a few hundred episodes, the learned policy always successfully guide to each sub-goal (convergence with continuous all light color), though some takes more episodes to train than others due to the environment layout (room connectivity). The right plot is a record for training Safe policy, where y-axis suggests cumulative ratio of successful episodes. Both plots would indicate a successful policy models learned with stable results, and the overall training takes within 1-2 minutes with an ordinary laptop device.

Figure 13 is the DFA rendered for the second task used for the Office grid experiment - delivering a coffee and a mail to the office (in any order). Blue dots marks all accessible paths from the starting state (1) to the only accepting state (4), and other edges are unsuccessful. Dijkstra path planning algorithms dynamically estimate the cost of traversing each edge and find a single path that is optimal.

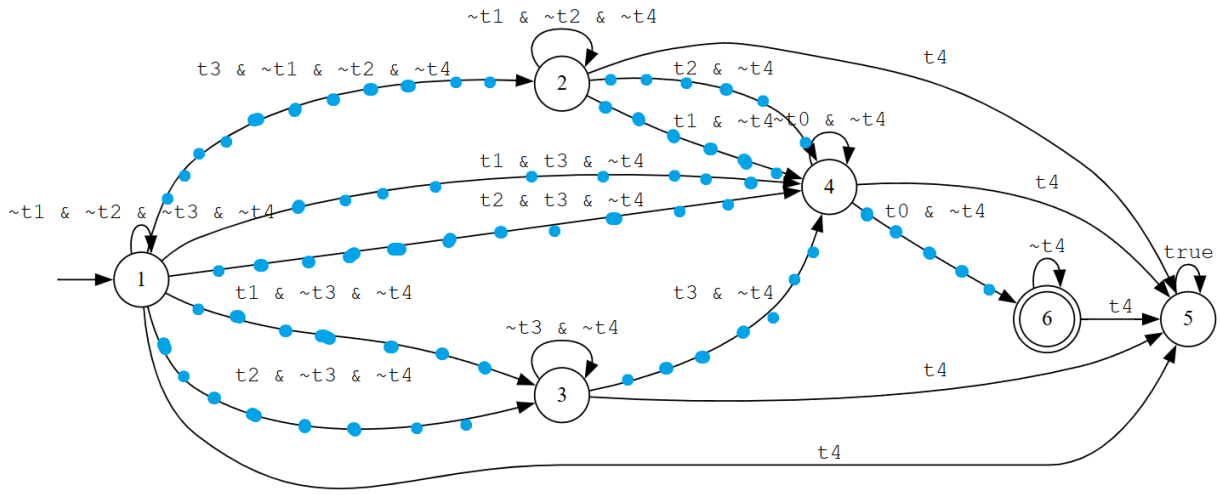

Figure 13: Office grid experiment complex task: DFA graph for "Delivering coffee and mail" (t1: $coffee_1$, t2: $coffee_2$, t3: *mail*, t4: obstacle, t0: $office$)