# Skill Machines: Temporal Logic Composition in Reinforcement Learning

Geraud Nangue Tasse[1], Devon Jarvis[1], Steven James[1], Benjamin Rosman[1]

***Abstract*—A major challenge in reinforcement learning is specifying tasks in a manner that is both interpretable and verifiable. One common approach is to specify tasks through reward machines—finite state machines that encode the task to be solved. We introduce skill machines, a representation that can be learned directly from these reward machines that encode the solution to such tasks. We propose a framework where an agent first learns a set of base skills in a reward-free setting, and then combines these skills with the learned skill machine to produce composite behaviours specified by any regular language, such as linear temporal logics. This provides the agent with the ability to map from complex logical task specifications to near-optimal behaviours zero-shot. We demonstrate our approach in both a tabular and high-dimensional video game environment, where an agent is faced with several of these complex, long-horizon tasks. Our results indicate that the agent is capable of satisfying extremely complex task specifications, producing near optimal performance with no further learning. Finally, we demonstrate that the performance of skill machines can be improved with regular off-policy reinforcement learning algorithms when optimal behaviours are desired.**

## I. Introduction

Reinforcement learning (RL) is a promising framework for developing truly general agents capable of acting autonomously in the real world. Despite recent successes in the field, ranging from video games [1] to robotics [2], there are several shortcomings to existing approaches that hinder RL's real-world applicability. One issue is that of sample efficiency—while it is possible to collect millions of data points in a simulated environment, it is simply not feasible to do so in the real world. This inefficiency is exacerbated when a single agent is required to solve multiple tasks (as we would expect of a generally intelligent agent). Another issue arises when an agent is required to solve a long horizon task in the presence of a sparse learning signal. In this case, it is often near impossible for the agent to solve the task, regardless of how much data it collects, since the sequence of actions to execute before a learning signal is received is too large [3]. However, this can be mitigated by leveraging higher-order skills, which shorten the planning horizon [4].

A desirable characteristic of generally intelligent agents is their ability to reuse learned behaviours to solve new tasks [5], preferably without further learning. One approach to overcoming this challenge is to rely on *composition*, where an agent first learns individual skills and then combines them to produce novel behaviours. There are several notions of compositionality in the literature, such as temporal composition, where skills are invoked one after the other [4], [6], and concurrent composition, where skills are combined to produce a new behaviour to be executed [7], [8], [9], [10].

Notably, recent work [11] has demonstrated how an agent can learn skills that can be combined using Boolean operators, such as negation and conjunction, to produce semantically meaningful behaviours without further learning. An important benefit of this compositional approach is that it provides a way to address another key issue with RL: tasks, as defined by reward functions, can be notoriously difficult to specify. This may lead to undesired behaviours that are not easily interpretable and verifiable. Composition that enables simpler task specifications and produces reliable behaviours thus represents a major step towards safe AI [12].

Unfortunately, these compositions are strictly concurrent and cannot be chained to solve temporally-specified tasks. One solution to this issue is *reward machines*—finite state machines that encode the tasks to solve [13]. While this obviates the sparse reward problem, the agent is still required to learn how to solve a given task through environment interaction, and the subsequent solution is monolithic, restricting its applicability to new tasks and limiting the reliability of resulting behaviours.

In this work, we combine these two approaches to develop an agent capable of zero-shot concurrent *and* temporal composition. We particularly focus on temporal logic composition, such as linear temporal logic (LTL) [14], allowing agents to sequentially chain and order their skills while ensuring certain conditions are always or never met. We make the following contributions: (a) we propose *skill machines*, a finite state machine that can be autonomously learned by a compositional agent, and which can be used to solve any task expressible as a finite state machine without further learning; (b) we prove that these skill machines are *satisficing*—given a task specification, an agent can successfully solve it while adhering to any constraints; and (c) we demonstrate our approach in several environments, including a high-dimensional video game domain. Having learned a set of base skills in a reward-free setting, our results indicate that our method is capable of producing near-optimal behaviour for a variety of long-horizon tasks without further learning.

## II. Background

We model the agent's interaction with the world as a Markov Decision Process (MDP), given by $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where (i) $\mathcal{S} \subseteq \mathbb{R}^n$ is the $n$-dimensional state space; (ii) $\mathcal{A}$ is the set of (possibly continuous) actions available to the agent; (iii) $P(s'|s,a)$ is the dynamics of the world, representing

[1]School of Computer Science and Applied Mathematics, University of the Witwatersrand, Johannesburg, South Africa geraudnt@gmail.com, {devon.jarvis, steven.james, benjamin.rosman1}@wits.ac.za

the probability of the agent reaching state $s'$ after executing action $a$ in state $s$; (iv) $R$ is a reward function bounded by $[R_{\text{MIN}}, R_{\text{MAX}}]$ that represents the task the agent needs to solve; and (v) $\gamma \in [0, 1]$ is a discount factor.

The aim of the agent is to compute a Markov policy $\pi$ from $\mathcal{S}$ to $\mathcal{A}$ that optimally solves a given task. Instead of directly learning a policy, an agent will often instead learn a value function that represents the expected return following policy $\pi$ from state $s$: $V^\pi(s) = \mathbb{E}^\pi\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$. A more useful form of value function is the action-value function $Q^\pi(s, a)$, which represents the expected return obtained by executing $a$ from $s$, and then following $\pi$. The optimal action-value function is given by $Q^*(s, a) = \max_\pi Q^\pi(s, a)$ for all states $s$ and actions $a$, and the optimal policy follows by acting greedily with respect to $Q^*$ at each state.

### A. Logical Composition in the Multitask Setting

We are interested in the multitask setting, where an agent is required to reach a set of goals in some goal space $\mathcal{G} \subseteq \mathcal{S}$. We assume that all tasks share the same state space, action space and dynamics, but differ in their reward functions. We model this setting by defining a background MDP $M_0 = \langle \mathcal{S}_0, \mathcal{A}_0, P_0, R_0 \rangle$ with its own state space, action space, transition dynamics and background reward function. Any individual task $\tau$ is then specified by a task-specific reward function $R_\tau$ that is non-zero only for states in $\mathcal{G}$. The reward function for the resulting MDP is then simply $R_0 + R_\tau$.

Recent work [11] consider the case where $R_\tau \in \{R_{\text{MIN}}, R_{\text{MAX}}\}$ and develop a framework that allows agents to apply the Boolean operations of conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$) over the space of tasks and value functions. This is achieved by first defining the goal-oriented reward function $\bar{R}$ which extends the task rewards $r$ to penalise an agent for achieving goals different from the one it wished to achieve:

$$\bar{R}(s, g, a) \coloneqq \begin{cases} \bar{R}_{\text{MIN}} & \text{if } g \neq s \text{ and } s \text{ is absorbing} \\ R(s, a) & \text{otherwise,} \end{cases} \tag{1}$$

where $\bar{R}_{\text{MIN}}$ is a large negative penalty that can be derived from the bounds of the reward function.

Using Equation 1, we can define the related goal-oriented value function as:

$$\bar{Q}(s, g, a) = \bar{R}(s, g, a) + \gamma \int_{\mathcal{S}} \bar{V}^{\bar{\pi}}(s', g) P_{(s,a)}(ds'), \tag{2}$$

where $\bar{V}^{\bar{\pi}}(s, g) = \mathbb{E}_{\bar{\pi}}\left[\sum_{t=0}^{\infty} \gamma^t \bar{R}(s_t, g, a_t)\right]$.

If a new task can be represented as the logical expression of previously learned tasks, then the optimal policy can immediately be obtained by composing the learned goal-oriented value functions using the same expression [11]. For example, the union ($\vee$), intersection ($\wedge$), and negation ($\neg$) of two goal-reaching tasks $A$ and $B$ can be solved as follows (we omit the value functions' parameters for readability):

$$\bar{Q}^*_{A \vee B} = \bar{Q}^*_A \vee \bar{Q}^*_B \coloneqq \max\{\bar{Q}^*_A, \bar{Q}^*_B\}$$

$$\bar{Q}^*_{A \wedge B} = \bar{Q}^*_A \wedge \bar{Q}^*_B \coloneqq \min\{\bar{Q}^*_A, \bar{Q}^*_B\}$$

$$\bar{Q}^*_{\neg A} = \neg \bar{Q}^*_A \coloneqq \left(\bar{Q}^*_{SUP} + \bar{Q}^*_{INF}\right) - \bar{Q}^*_A,$$

where $\bar{Q}^*_{SUP}$ and $\bar{Q}^*_{INF}$ are the goal-oriented value functions for the maximum task ($r = R_{\text{MAX}}$ for all $\mathcal{G}$) and minimum task ($r = R_{\text{MIN}}$ for all $\mathcal{G}$), respectively . Following recent work [15], we refer to these goal-oriented value functions as *world value functions* (WVFs).

### B. Reward Machines

One difficulty with the standard MDP formulation is that the agent is often required to solve a complex long-horizon task using only a scalar reward signal as feedback from which to learn. To overcome this, recent work [13] propose *reward machines* (RMs), which provide structured feedback to the agent in the form of a finite state machine. RMs encode a reward function using a set of propositional symbols $\mathcal{P}$ that represent abstract environment features as follows:

**Definition 1** (Reward Machine)**.** *Given a set of propositional symbols $\mathcal{P}$, states $\mathcal{S}$ and actions $\mathcal{A}$, a reward machine is a tuple $R_{\mathcal{PSA}} = \langle U, u_0, F, \delta_u, \delta_r \rangle$ where (i) $U$ is a finite set of states; (ii) $u_0 \in U$ is an initial state; (iii) $F$ is a finite set of terminal states; (iv) $\delta_u : U \times 2^{\mathcal{P}} \to U \cup F$ is the state-transition function; and (v) $\delta_r : U \to [\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}]$ is the state-reward function.*

RMs consist of a finite set of states $U$, each of which represents a set of propositions that are true at a given environment state. Transitions between RM states are governed by $\delta_u$, and the RM emits a reward function according to $\delta_r$. A particular instantiation of an RM that is used in practice is a *simple reward machine* (SRM), which restricts the form of the state-reward function to be $\delta_r : U \times 2^{\mathcal{P}} \to \mathcal{R}$ [13]. In other words, when a transition between $u, u' \in U$ is made, the SRM emits a single scalar instead of a function (as in the case of RMs).

To incorporate RMs into the RL framework, the agent must be able to determine which abstract propositions are true at any given state. To achieve this, the agent is equipped with a labelling function $L : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to 2^{\mathcal{P}}$ that assigns truth values to the propositions based on the agent's interaction with its environment. The agent can then learn a policy in a new decision process where the reward function in the original MDP is replaced with the RM, which is defined by the tuple $\langle \mathcal{S}, \mathcal{A}, P, \gamma, \mathcal{P}, L, U, u_0, F, \delta_u, \delta_r \rangle$. The agent's aim is now to learn a policy over the joint MDP and RM state space $\pi : \mathcal{S} \times U \to \mathcal{A}$, which can be achieved with standard algorithms such as $Q$-learning [13].

## III. Leveraging Skill Composition for Temporal Logic Tasks

To describe our approach to temporal composition, we use the *Office Gridworld* [13] as a running example. In the environment, illustrated by Figure 1a, an agent (blue circle) can move to adjacent cells in any of the cardinal directions.

It can also pick up coffee or mail at locations ☕ or ✉ respectively, and it can deliver them to the office at location 🧍. Cells marked ✻ indicate decorations that are broken if the agent collides with them, and cells marked $A$–$D$ indicate the centres of the corner rooms. The reward machines that specify tasks in this environment are defined over 10 propositions: $\mathcal{P} = \{A, B, C, D, ✻, ☕, ✉, 🧍, ✉^+, 🧍^+\}$, where the first 8 propositions are true when the agent is at their respective locations, $✉^+$ is true when the agent is at ✉ and there is mail to be collected, and $🧍^+$ is true when the agent is at 🧍 and there is someone in the office.

### A. Task Space

We now define the set of tasks to be considered. We first introduce the concept of *constraints* $C \subseteq \mathcal{P}$, which are the set of propositions that an agent should avoid setting to true and corresponds to the global operator $G$ in a linear temporal logic (LTL) specification. An example of a constraint might be that the agent should complete a task, but avoid breaking any decorations while doing so. We can now define the notion of base (primitive) tasks, which will later be composed.

**Definition 2** (Task Primitive)**.** *Let $M = \langle \mathcal{S}_0, \mathcal{A}_0, P_0, R_0 \rangle$ be a background MDP. We define a set of task primitives in this domain as $\mathcal{M}_\mathcal{G} = \{\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle\}$ with absorbing goal space $\mathcal{G} = 2^\mathcal{P}$ and labelling function $L$, where*

$$\mathcal{S} \coloneqq (\mathcal{S}_0 \times 2^C) \cup 2^\mathcal{P}, \quad \mathcal{A} \coloneqq \mathcal{A}_0 \times \mathcal{A}_\tau;$$

$$P(\langle s_0, c \rangle, \langle a_0, a_\tau \rangle) \coloneqq \begin{cases} L(s_0, a_0, s_0') & \text{if } a_\tau = 1 \\ \langle s_0', c' \rangle & \text{otherwise} \end{cases},$$

$$R(\langle s_0, c \rangle, \langle a_0, a_\tau \rangle) \coloneqq \begin{cases} R_\tau \in \{R_{MIN}, R_{MAX}\} & \text{if } a_\tau = 1 \\ R_0(s_0, a_0) & \text{otherwise.} \end{cases}$$

*where $C$ is the set of constraints, $\mathcal{A}_\tau = \{0, 1\}$ represents whether or not to terminate a task, $s_0' \sim P_0(\cdot | s_0, a_0)$ and $c' = c \cup (c \cap L(s_0, a_0, s_0'))$.*

The above defines the tasks' state space to be the product of the environment state and the set of constraints, incorporating the set of propositions that are currently true. The action space is augmented with a terminating action following previous works [6], [11], which indicates that the agent wishes to achieve the goal it is currently at, and is similar to an option's termination condition [4]. The transition dynamics update the environment state and constraints set to true when a regular action is taken, and use the labelling function to return the set of propositions achieved when the agent decides to terminate. Finally, the agent receives the regular environment reward when taking an action, but a task-specific goal reward when it terminates. We will assume that the environment and task-specific rewards are such that the optimal policies for all tasks are guaranteed to attain desired reachable goal states—a common example is to have a reward of 1 at desired goal states and no rewards everywhere else.

Equipped with this definition, we can now define the set of all tasks under consideration:

**Definition 3** (Task space)**.** *The set of all tasks $\mathcal{M}$ is all linear preferences over task primitives:*

$$\mathcal{M} = \Big\{ \langle \mathcal{S}, \mathcal{A}, P, R_\mathbf{w}, \gamma \rangle : R_\mathbf{w}(s, a) = \sum_{i=1}^{|\mathcal{M}_\mathcal{G}|} w_i R_i(s, a) \\ \text{where } \sum_{i=1}^{|\mathcal{M}_\mathcal{G}|} w_i = 1 \text{ and } \mathbf{w} \in \mathbb{R}^{|\mathcal{M}_\mathcal{G}|} \Big\}$$

This definition of task space provides a general notion of tasks that are still grounded in achieving goals. It also subsumes most definitions considered in the composition literature, including Boolean algebra tasks [11] and linear preference tasks [16]. Hence, we will restrict our attention to reward machines whose rewards per state originate from the defined task space (instead of arbitrary real-valued functions that are not grounded in achieving goals in an environment). We denote such reward machines as $R_{\mathcal{MSA}} = \langle U, u_0, F, \delta_u, \delta_\mathcal{M} \rangle$ where $\delta_\mathcal{M}$ maps from RM states to task rewards.

### B. Skill Machines

The goal space of a primitive task is defined by a set of Boolean propositions. We can leverage prior work to solve each individual task using a set of base primitive skills [11]. We therefore only need concern ourselves with how to solve any task expressed as a linear combination of the primitive tasks. Fortunately, Theorem 1 below demonstrates that a linear combination of base skills does just this (proofs of all theorems are presented in the Appendix).

**Theorem 1.** *Let $\mathbf{R}_{\mathcal{M}_\mathcal{G}}$ be a vector of rewards for each primitive task, and $\bar{\mathbf{Q}}^*_\mathcal{G}$ be the corresponding vector of optimal WVFs. Then, for a task $m \in \mathcal{M}$ with reward function $R_\mathbf{w} = \mathbf{w} \cdot \mathbf{R}_{\mathcal{M}_\mathcal{G}}$, we have*

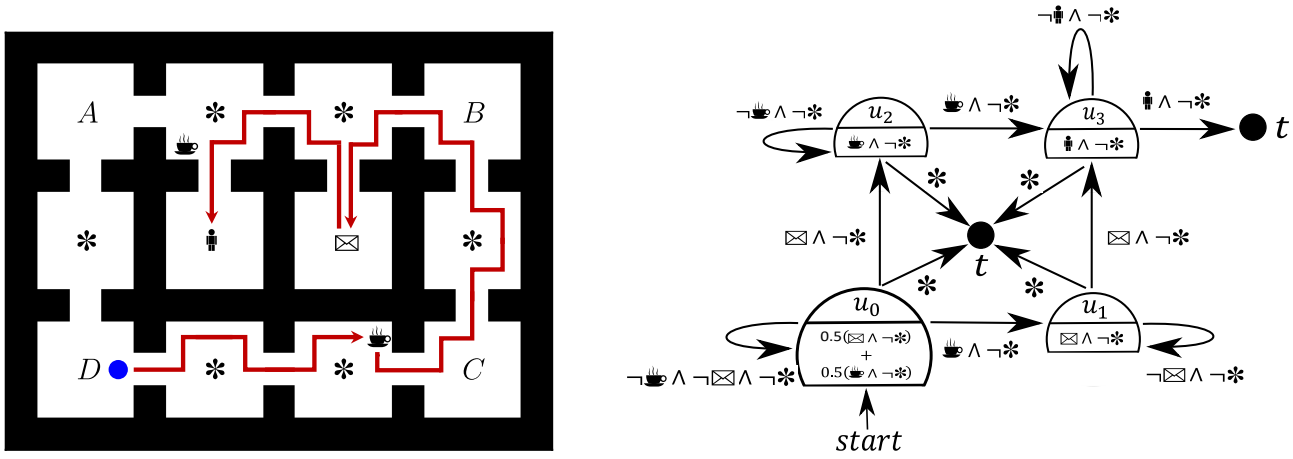
$$\bar{Q}^*_m = \mathbf{w} \cdot \bar{\mathbf{Q}}^*_\mathcal{G}.$$

*Proof.*

$$\begin{aligned} \bar{Q}^*_m(s, g, a) &= \mathbb{E}_{\bar{\pi}^*} \left[ \sum_{t=0}^{\infty} \gamma^t \mathbf{w} \cdot \bar{\mathbf{R}}_{\mathcal{M}_\mathcal{G}}(s_t, g, a_t) \right] \\ &= \mathbf{w} \cdot \mathbb{E}_{\bar{\pi}^*} \left[ \sum_{t=0}^{\infty} \gamma^t \bar{\mathbf{R}}_{\mathcal{M}_\mathcal{G}}(s_t, g, a_t) \right]; \text{ since the} \\ &\quad \text{policies are independent of task [11][Lem 2].} \\ &= \mathbf{w} \cdot \bar{\mathbf{Q}}^*_\mathcal{G} \end{aligned}$$

□

We now have agents capable of solving any logical and linear composition of tasks by learning a finite set of base skills $\bar{\mathbf{Q}}^*_\mathcal{G}$ for task primitives $\mathcal{M}_\mathcal{G}$. We refer to this basis set of skills as *skill primitives*. Given this compositional ability over skills, and reward machines that expose the structure of tasks, agents can solve temporally extended tasks with little or no further learning. To achieve this, we define a skill machine (SM) as a representation of logical and temporal knowledge over skills.




(a) Office Gridworld (b) Skill machine

Fig. 1: Illustration of (a) the office gridworld where the blue circle represents the agent and (b) the skill machine for the task "deliver coffee and mail to the office without breaking any decoration" where the black dots labeled $t$ represent terminal states.

**Definition 4** (Skill Machine)**.** *Given a set of propositional symbols $\mathcal{P}$ with constraints $\mathcal{C} \subseteq \mathcal{P}$, their corresponding skill primitives $\bar{\mathbf{Q}}^*_{\mathcal{G}}$ and task space $\mathcal{M}$, states $\mathcal{S}$ and actions $\mathcal{A}$, a skill machine is a tuple $\bar{\mathcal{Q}}^*_{\mathcal{MSA}} = \langle U, u_0, F, \delta_u, \delta_Q, \mathbf{w}_U, \mathbf{w}_{\mathcal{G}} \rangle$ where (i) $U$, $u_0$, $F$, $\delta_u$ are defined as for reward machines; (ii) $\mathbf{w}_U : U \times U \to \mathbb{R}$ is a preference function over state transitions; (iii) $\mathbf{w}_{\mathcal{G}} : \mathcal{S} \times \mathcal{G} \to \mathbb{R}$ is a preference function over goals; and (v) $\delta_Q : \mathcal{S} \times U \times \mathcal{A} \to [\mathcal{S} \times \mathcal{A} \to \mathbb{R}]$ is the state-skill function defined by:*

$$\delta_Q(s,u,a) \mapsto \sum_{g \in \mathcal{G}} \sum_{u' \in U} \mathbf{w}_{\mathcal{G}}(s,u,g) \mathbf{w}_U(u,u') \bar{Q}^*_{u,u'}(s,g,a),$$

*where $\bar{Q}^*_{u,u'}$ is the WVF obtained by composing the skill primitives $\bar{\mathbf{Q}}^*_{\mathcal{G}}$ according to the Boolean expression for the transition $\delta_u = u'$.*

For a given state $s$ in the environment and state $u$ in the skill machine, the skill machine uses its preference over transitions $\mathbf{w}_U$ and goals $\mathbf{w}_{\mathcal{G}}$ to compute a skill $Q(s,a) := \delta_Q(s,u,a)$ that an agent can use to take an action $a$. The environment then transitions to the next state $s' \leftarrow P(s,a)$ and the skill machine transitions to $u' \leftarrow \delta_u(u, L(s,a,s'))$. $\mathbf{w}_U$ represents cases where there is not necessarily a single desirable transition to follow given the current SM state. This is illustrated by the SM in Figure 1b, where mail and coffee are equally desirable at the initial state. Similarly, $\mathbf{w}_{\mathcal{G}}$ represents cases where there may be a single desirable task, but its goals are not necessarily equally desirable given the environment state—for example when the agent needs to first pick up coffee but there are two coffee locations. Interestingly, there always exists a choice for $\mathbf{w}_U$ and $\mathbf{w}_{\mathcal{G}}$ that is optimal with respect to the corresponding reward machine, as shown in Theorem 2.

**Theorem 2.** *Let $\pi^*(s,u)$ be the optimal policy for the cross-product MDP between a reward machine and a task space $\mathcal{M}$, with $\mathcal{C} = \mathcal{P}$. Then there exists a corresponding skill machine with a $\mathbf{w}_{\mathcal{G}}$ and $\mathbf{w}_U$ such that*

$$\pi^*(s,u,a) \in \arg\max_a \delta_Q(s,u,a),$$

*where $\delta_Q$ is given by $\mathbf{w}_{\mathcal{G}}$ and $\mathbf{w}_U$ as per Definition 4.*

*Proof.* Let $\mathbf{w}_U(u,\cdot) = \frac{1}{N_{\delta_u}}$ where $N_{\delta_u}$ is the number of possible RM transitions from $u$. Also let $\mathbf{w}_{\mathcal{G}}(s,u,\cdot)$ be 1 for the set of propositions $g \in 2^{\mathcal{C}}$ that are satisfied when following $\pi^*(s,u)$, and zero everywhere else. Then $\pi^*(s,u) \in \arg\max_a \delta_Q(s,u)$ since $\mathbf{w}_U(u,u')\bar{Q}^*_{u,u'}(s,g,a)$ is optimal using Theorem 1 and optimal policies are assumed to reach task goals. □

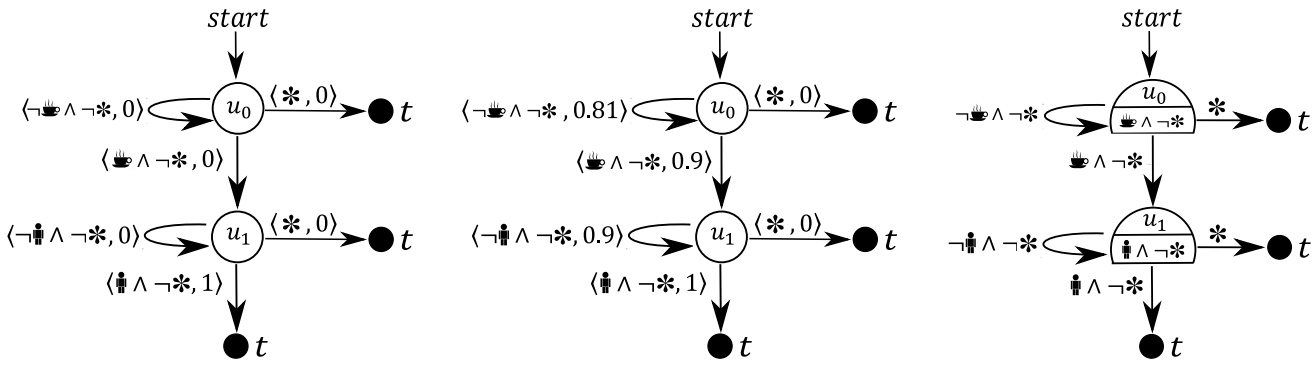
Theorem 2 shows that skill machines can be used to solve tasks without having to relearn action level policies. The next section shows how an agent can approximate a skill machine by planning over simple reward machines.

### C. From Reward Machines to Skill Machines

In the previous section, we introduced skill machines and showed that they can be used to represent the logical and temporal composition of skills needed to solve reward machines. We now show how for simple RMs their approximate SM can be obtained zero-shot without further learning. To achieve this, we first plan over the reward machine (using value iteration, for example) to obtain Q-values for each transition. We then select the skills for each SM state greedily. This process is illustrated in Figure 2. While this only holds for cases where the greedy skills are always satisfiable from any environment state, this still covers many tasks of interest. In particular, this holds for any RM with non-zero rewards of $R_{\text{MAX}}$ at accepting transitions,[1] as shown in Theorem 3.

**Theorem 3.** *Let $R_{\mathcal{MSA}} = \langle U, u_0, F, \delta_u, \delta_{\mathcal{M}} \rangle$ be a satisfiable simple reward machine with non-zero rewards $R_{MAX}$ only for*

[1]Accepting transitions are transitions at which the high level task—described, for example, by linear temporal logics—is satisfied.




(a) Reward machine (b) Value iterated RM (c) Skill machine

Fig. 2: The reward machine, value iterated RM and skill machine for the task "Deliver coffee to the office without breaking any decoration". This task is specified using LTL as $(F(\text{☕} \wedge X(F\ \text{🧍}))) \wedge (G\ \neg \text{✻})$, where $F = Finally, X = neXt, G = Globally$ are LTL operators. The corresponding RM is obtained by converting the LTL into a finite state machine and then giving a reward of 1 for accepting transitions and 0 otherwise. The black dots labeled $t$ represent terminal states.

*accepting transitions, and for which all valid transitions* $(u, u')$ *are achievable from any state* $s \in \mathcal{S}$*. Define the skill machine* $\bar{\mathcal{Q}}^*_{\mathcal{MSA}} = \langle U, u_0, F, \delta_u, \delta_Q, \mathbf{w}_U, \mathbf{w}_\mathcal{G} \rangle$ *with*

$$\mathbf{w}_U(u, u') \coloneqq \begin{cases} 1 & \text{if } u' = \arg\max_{u''} Q^*(u, u''), \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{w}_\mathcal{G}(s, u, g) \coloneqq \begin{cases} 1 & \text{if } g = \arg\max_{g'} \max_a \sum_{u'} \mathbf{w}_U \bar{Q}^*_{u,u'}{}^2, \\ 0 & \text{otherwise} \end{cases}$$

*where* $Q^*$ *is the optimal transition-value function for* $R_{\mathcal{PSA}}$*. Then following the policy* $\pi^*(s, u) \in \arg\max_a \delta_Q(s, u, a)$ *will reach an accepting transition.*

*Proof.* This follows from the optimality of $\pi^*(s, u)$ and $Q^*$, since each transition of the RM is satisfiable from any environment state. □

Theorem 3 is critical as it provides soundness guarantees, ensuring that the policy derived from the skill machine will always satisfy the task requirements when it is possible to do so. Finally, in cases where the composed skill $\delta_Q$ obtained from the approximate SM is not sufficiently optimal, we can use any off-policy algorithm to learn a new skill $Q_{new}$ few-shot. This is achieved by using the maximising Q-values $\max\{\beta Q_{new}, (1-\beta)\delta_Q\}$ in the behaviour policy during learning. Here, $\beta \in [0, 1]$ is a parameter that determines how much of the composed policy to use. It can also be seen as decreasing the potentially overestimated values of $\delta_Q$, since $\delta_Q$ is greedy with respect to both goals and RM transitions. Algorithm 1 illustrates this process with Q-learning where $\beta = \gamma$, which guarantees convergence since $\delta_Q$ will never dominate the optimal Q-values in the $\max$.

**Algorithm 1:** Few-shot Q-learning using skill machines

**Input :** $\gamma, \alpha, \mathcal{P}, L, U, u_0, F, \delta_u, \delta_Q$
**Initialise :** $Q(s, u, a)$
**foreach** *episode* **do**
  Observe initial state $s \in \mathcal{S}$ and get initial $u \leftarrow u_0$
  **while** *episode is not done* **do**
    `/* Using the composed skill` $\delta_Q$ `in the behaviour policy */`
    $a \leftarrow \arg\max_{a \in \mathcal{A}} (\max\{\gamma Q(s, u, a), (1-\gamma)\delta_Q(s, u, a)\})$
    **if** Bernoulli(1-$\epsilon$) $= 1$ **else** a random action
    Take action $a$ and observe next state $s'$
    Get reward $r \leftarrow \delta_r(u)(s, a, s')$ and the next RM state $u' \leftarrow \delta_u(u, L(s, a, s'))$
    $Q(s, u, a) \xleftarrow{\alpha} r$ **if** $s'$ is terminal or $u' \in F$ **else** $\left[r + \gamma \max_{a'} Q(s', u', a')\right]$
    $s \leftarrow s'$

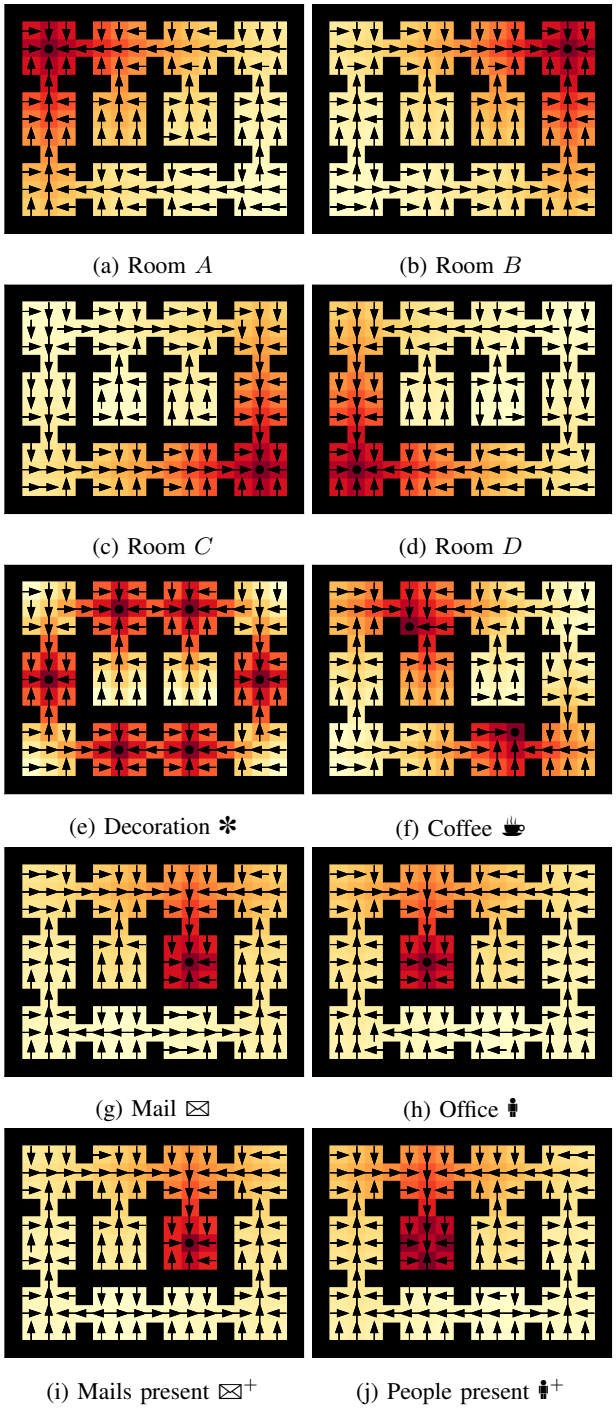
| Task | Description |
|---|---|
| 1 | Deliver coffee to the office without breaking any decoration |
| 2 | Patrol rooms A, B, C, and D without breaking any decoration |
| 3 | Deliver coffee and mail to the office without breaking any decoration |
| 4 | Deliver mail to the office until there is no mail left, then deliver coffee to office while there are people in the office, then patrol rooms A-B-C-D-A, and never break a decoration |

TABLE I: Tasks in the Office Gridworld.

## IV. Experiments

### A. Zero-shot temporal logics

We consider the Office Gridworld domain [13] depicted in Figure 1a. This environment is used as a multitask domain, consisting of the four tasks described in Table I. We begin by training the agent on all 10 base tasks of the Office Gridworld



(a) Room $A$ (b) Room $B$

(c) Room $C$ (d) Room $D$

(e) Decoration ✻ (f) Coffee

(g) Mail ✉ (h) Office

(i) Mails present $✉^{+}$ (j) People present $^{+}$

Fig. 3: The policies (arrows) and value functions (heat map) of the base primitive tasks in the Office Gridworld. These are obtained by maximising over the goals of the learned WVFs. All errors in the figures are due to training the WVFs for 200000 time steps, hence not to convergence.

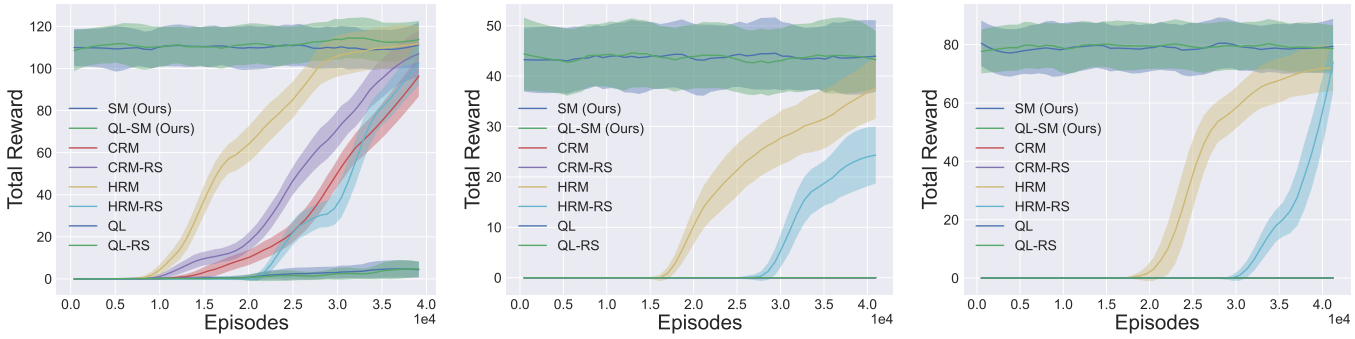
(Figure 3) and then evaluating how long it takes it to learn a policy that can solve all four tasks. The agent iterates through the tasks, changing from one to the next after each episode. In all of our experiments, we compare the performance of skill machines with that of state-of-the-art RM-based learning approaches like counterfactual RMs (CRM)—where the Q-functions are updated with respect to all possible RM transitions from a given environment state—and hierarchical RMs (HRM)—where an agent learns options per RM state that are grounded in the environment states [13]. Note that CRM and HRM are theoretically capable of solving the multi-task problem setup because they can use experience from solving one task to update the policies for solving other tasks. However, skill machines can additionally share both experience between tasks when learning the skill primitives, as well as use the composition of these primitives to generalise to more difficult tasks without further learning.

We run 80 independent trials and report the average reward per step across the four tasks in Table I. In addition to learning all four tasks, we also experiment with Tasks 3 and 4 in isolation. For these experiments, 80 independent trials are run and the average reward per step computed. In the single task domains, the difference between CRM, HRM, skill machines and Q-learning should be less pronounced, since CRM, HRM and skill machines now cannot leverage prior knowledge. Thus, the comparison between multi-task and single-task learning in this setting will evaluate the benefit of the compositionality afforded by skill machines.

The results of these three experiments are shown in Figure 4. Regular Q-learning struggles to learn Task 3 and completely fails to learn the hardest task (Task 4). Additionally, notice that while QL and CRM can theoretically learn the tasks optimally given infinite time, only HRM and SM are able to learn hard long horizon tasks in practice. It is important to note that we train all algorithms for the same amount of time during these experiments and previous work [11] has shown that learning the WVFs takes longer than learning task-specific skills. In addition, the skill machines are being used to zero-shot generalise to the office tasks using skill primitives. Thus using the skill machines in isolation (labelled SM and shown in blue on Figure 4) may provide sub-optimal performance compared to the task-specific agents, since the skill machines have not been trained to optimality and are not specialised to the domain. Even under these conditions, we observe that skill machines perform near-optimally in terms of final performance, and due to the amortised nature of learning the WVF will achieve its final rewards from the first epoch.

### *B. Few-shot Temporal Logics*

It is possible to pair the skill machines with a learning algorithm such as Q-learning to achieve few-shot generalisation. From the results shown in Figure 4, it is apparent that skill machines paired with Q-learning (labelled QL-SM and shown in orange on Figure 4) achieves the best performance for both the single-task and multi-task setting. While it is not clear from the rewards that adding Q-learning provides significant improvements to the skill machine, their trajectories show that Q-learning does indeed improve on the skill machine policies when they are not optimal (Figure 5). Additionally, skill machines with Q-learning always begin with a significantly




(a) Task 3 (b) Task 4 (c) Multiple tasks (1-4)

Fig. 4: Average returns during training in the Office Gridworld.

higher reward and converge on their final performance faster than all benchmarks—except the zero-shot one which is (near) optimal in all cases. The speed of learning is due to the compositionality of the skill primitives with skill machines, and the high final performance is due to the generality of the learned primitives being paired with the domain specific Q-learner. In sum, skill machines provide fast composition of skills and achieve optimal performance compared to all benchmarks when paired with a learning algorithm.

### C. Moving Targets Domain

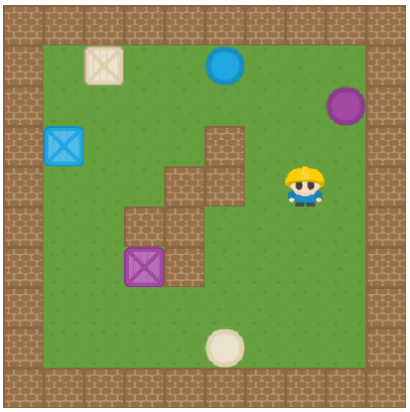


Fig. 6: Moving Targets domain

| Task | Description |
|---|---|
| 1 | Pick up any object. Repeat this forever. |
| 2 | Pick up blue then purple objects, then objects that are neither blue nor purple. Repeat this forever. |
| 3 | Pick up blue objects or squares, but never blue squares. Repeat this forever. |
| 4 | Pick up non-square blue objects, then non-blue squares in that order. Repeat this forever. |

TABLE II: Tasks in the Moving Targets domain. Objects respawn in random positions when picked.

We now demonstrate our temporal logic composition approach in a canonical object collection domain with high dimensional pixel observations [11] (Figure 6). The agent here needs to pick up objects of various shapes and colors; picked objects respawn at random empty positions similarly to previous object collection domains [16]. There are 3 object colours—beige, blue, purple—and 2 object shapes—squares, circles. To learn the WVF for a given task primitive, we use goal oriented Q-learning [11] where the agent keeps track of reached goals and uses deep Q-learning [17] to update the WVF with respect to all seen goals at every time step.

We first train the agent on three base task primitives: *pick up blue objects*, *pick up purple objects*, and *pick up squares*. We then use the learned skill primitives to solve multiple temporal logic tasks. Figure 7 shows the average returns of the optimal policies and SM policies for the four tasks described in Table II with a maximum of 50 steps per episode. Our results show that even when using function approximation with sub-optimal skill primitives, the zero-shot policies obtained from skill machines are very close to optimal on average. We also observe that for very challenging tasks like Tasks 3 and 4 (where the agent must satisfy difficult temporal constraints), the compounding effect of the sub-optimal policies sometimes leads to failures. In such cases, learning new skills few-shot by leveraging the SM would guarantee convergence to optimal policies as demonstrated in Section IV-B.

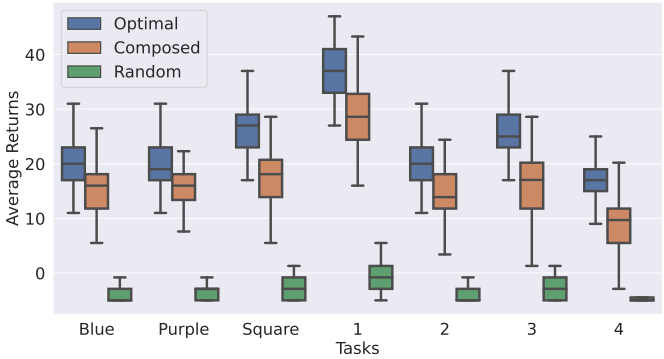
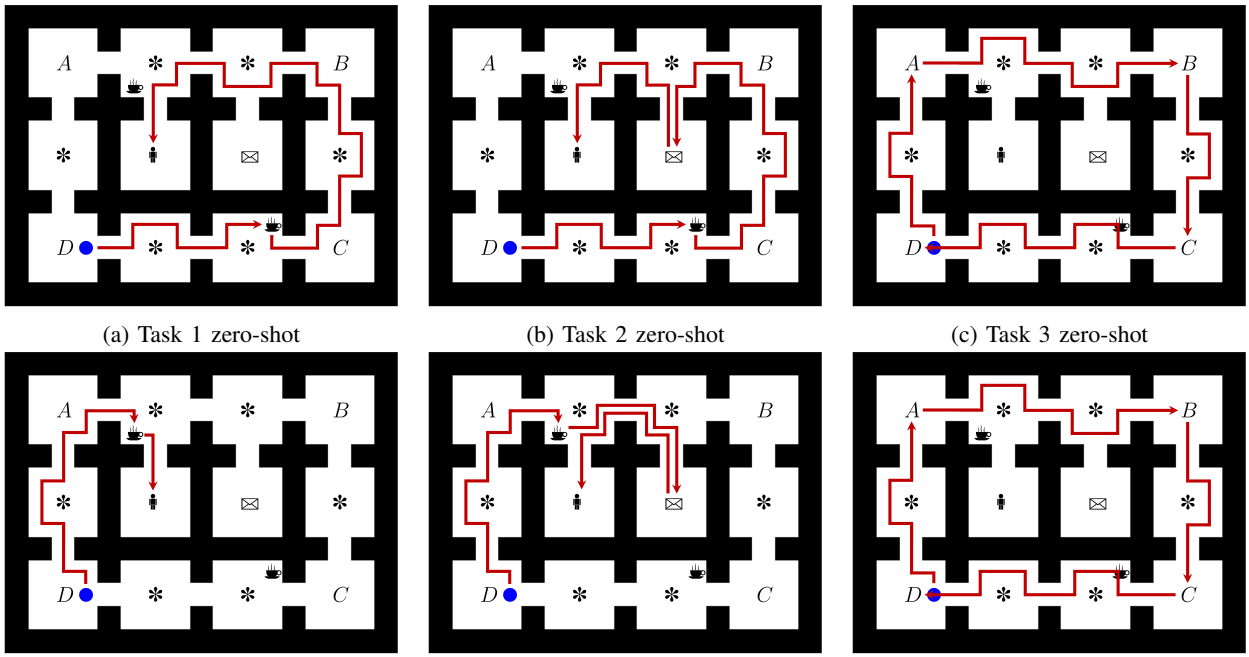



Fig. 7: Average returns over 100 runs for tasks in Table II.




(a) Task 1 zero-shot (b) Task 2 zero-shot (c) Task 3 zero-shot

(d) Task 1 few-shot (e) Task 2 few-shot (f) Task 3 few-shot

Fig. 5: Agent trajectories using the skill machine without further learning (top) and with further learning (bottom).

## V. Related Work

One family of approaches to concurrent composition leverages forms of regularisation to achieve semantically meaningful disjunction [7], [9] or conjunction [18], [19]. Weighted composition has also been demonstrated; for example, previous work [20] learn weights to compose existing policies multiplicatively to solve new tasks. Approaches that leverage the successor feature (SF) framework [21] are capable of solving tasks defined by linear preferences over features [16]. Recent work [10] shows that an SF basis can be learned that is sufficient to span the space of tasks under consideration, and one can also determine which policies should be stored in limited memory so as to maximise performance on future tasks [22]. In contrast to these approaches, our framework allows for both concurrent composition (with operators such as negation that other approaches do not support) and temporal composition such as LTL.

A popular way of achieving temporal composition is through the options framework [4], [23]. Here, high-level skills are first discovered and then executed sequentially to solve a task [24], [25]. One can leverage the SF and options framework and learn how to linearly combine skills, chaining them sequentially to solve temporal tasks [6]. However, these options-based approaches offer a relatively simple form of temporal composition. By contrast, we are able to solve tasks expressed through regular languages zero-shot, while providing soundness guarantees.

Work has also centred on approaches to defining tasks using human-readable logic operators. For example, one can specify tasks using LTL, which is then used to generate a standard reward signal for an RL agent [26], [27]. Some recent works [28] show how to perform reward shaping given LTL specifications, while others [29] develop a formal language that encodes tasks as sequences, conjunctions and disjunctions of subtasks. This is then used to obtain a shaped reward function that can be used for learning. All of these approaches focus on how an agent can improve learning given such specifications or structure, but we show how an explicitly compositional agent can immediately solve such tasks using WVFs without further learning.

## VI. Conclusion

We proposed skill machines—finite state machines that can be learned from reward machines—that allow agents to solve extremely complex tasks involving temporal and concurrent composition. We demonstrated how skills can be learned and encoded in a specific form of goal-oriented value function that, when combined with the learned skill machines, are sufficient for solving subsequent tasks without further learning. Our approach guarantees that the resulting policy adheres to the logical task specification, which provides assurances of safety and verifiability to the agent's decision making, important characteristics that are necessary if we are to ever deploy RL agents in the real world. While the resulting behaviour is provably satisficing, empirical results demonstrate that the agent's performance is near optimal; further fine-tuning can be performed should optimality be required, which greatly improves the sample efficiency. We see this approach as a step towards truly generally intelligent agents, capable of immediately solving human-specifiable tasks in the real world with no further learning.

## References


[1] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. D. Guo, and C. Blundell, "Agent57: Outperforming the Atari human benchmark," in *International Conference on Machine Learning*, 2020, pp. 507–517.

[2] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[3] J. Arjona-Medina, M. Gillhofer, M. Widrich, T. Unterthiner, J. Brandstetter, and S. Hochreiter, "Rudder: Return decomposition for delayed rewards," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[4] R. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial Intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[5] M. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: a survey," *Journal of Machine Learning Research*, vol. 10, pp. 1633–1685, 2009.

[6] A. Barreto, D. Borsa, S. Hou, G. Comanici, E. Aygün, P. Hamel, D. Toyama, S. Mourad, D. Silver, D. Precup *et al.*, "The option keyboard: Combining skills in reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[7] E. Todorov, "Compositionality of optimal control laws," in *Advances in Neural Information Processing Systems*, 2009, pp. 1856–1864.

[8] A. Saxe, A. Earle, and B. Rosman, "Hierarchy through composition with multitask LMDPs," *International Conference on Machine Learning*, pp. 3017–3026, 2017.

[9] B. Van Niekerk, S. James, A. Earle, and B. Rosman, "Composing value functions in reinforcement learning," in *International Conference on Machine Learning*, 2019, pp. 6401–6409.

[10] S. Alver and D. Precup, "Constructing a good behavior basis for transfer using generalized policy updates," in *International Conference on Learning Representations*, 2022.

[11] G. Nangue Tasse, S. James, and B. Rosman, "A Boolean task algebra for reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9497–9507, 2020.

[12] V. Cohen, G. Nangue Tasse, N. Gopalan, S. James, M. Gombolay, and B. Rosman, "Learning to follow language instructions with compositional policies," in *AAAI Fall Symposium Series*, 2021.

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

[14] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science*. IEEE, 1977, pp. 46–57.

[15] G. Nangue Tasse, S. James, and B. Rosman, "World value functions: Knowledge representation for multitask reinforcement learning," in *The 5th Multi-disciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, 2022.

[16] A. Barreto, S. Hou, D. Borsa, D. Silver, and D. Precup, "Fast reinforcement learning with generalized policy updates," *Proceedings of the National Academy of Sciences*, vol. 117, no. 48, pp. 30 079–30 087, 2020.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[18] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, and S. Levine, "Composable deep reinforcement learning for robotic manipulation," in *2018 IEEE International Conference on Robotics and Automation*, 2018, pp. 6244–6251.

[19] J. Hunt, A. Barreto, T. Lillicrap, and N. Heess, "Composing entropic policies using divergence correction," in *International Conference on Machine Learning*, 2019, pp. 2911–2920.

[20] X. B. Peng, M. Chang, G. Zhang, P. Abbeel, and S. Levine, "MCP: Learning composable hierarchical control with multiplicative compositional policies," in *Advances in Neural Information Processing Systems*, 2019, pp. 3686–3697.

[21] A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul, H. P. van Hasselt, and D. Silver, "Successor features for transfer in reinforcement learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4055–4065.

[22] M. Nemecek and R. Parr, "Policy caches with successor features," in *International Conference on Machine Learning*, 2021, pp. 8025–8033.

[23] P. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[24] G. Konidaris and A. Barto, "Skill discovery in continuous reinforcement learning domains using skill chaining," *Advances in Neural Information Processing Systems*, vol. 22, 2009.

[25] A. Bagaria and G. Konidaris, "Option discovery using deep skill chaining," in *International Conference on Learning Representations*, 2019.

[26] X. Li, C.-I. Vasile, and C. Belta, "Reinforcement learning with temporal logic rewards," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017, pp. 3834–3839.

[27] M. Littman, U. Topcu, J. Fu, C. Isbell, M. Wen, and J. MacGlashan, "Environment-independent task specifications via GLTL," *arXiv preprint arXiv:1704.04341*, 2017.

[28] A. Camacho, R. T. Icarte, T. Klassen, R. Valenzano, and S. McIlraith, "LTL and beyond: Formal languages for reward function specification in reinforcement learning." in *International Joint Conferences on Artificial Intelligence*, vol. 19, 2019, pp. 6065–6073.

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