

# Learning Reward Machines in Cooperative Multi-Agent Tasks

Leo Ardon  
Imperial College London

Daniel Furelos-Blanco  
Imperial College London

Alessandra Russo  
Imperial College London

## ABSTRACT

This paper presents a novel approach to Multi-Agent Reinforcement Learning (MARL) that combines cooperative task decomposition with the learning of reward machines (RMs) encoding the structure of the sub-tasks. The proposed method helps deal with the non-Markovian nature of the rewards in partially observable environments and improves the interpretability of the learnt policies required to complete the cooperative task. The RMs associated with each sub-task are learnt in a decentralised manner and then used to guide the behaviour of each agent. By doing so, the complexity of a cooperative multi-agent problem is reduced, allowing for more effective learning. The results suggest that our approach is a promising direction for future research in MARL, especially in complex environments with large state spaces and multiple agents.

## KEYWORDS

Reinforcement Learning; Multi-agent; Reward Machine; Neuro-Symbolic

### ACM Reference Format:

Leo Ardon, Daniel Furelos-Blanco, and Alessandra Russo. 2023. Learning Reward Machines in Cooperative Multi-Agent Tasks. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 8 pages.

## 1 INTRODUCTION

With impressive advances in the past decade, the Reinforcement Learning (RL) [18] paradigm appears as a promising avenue in the quest to have machines learn autonomously how to achieve a goal. Originally evaluated in the context of a single agent interacting with the rest of the world, researchers have since then extended the approach to the multi-agent setting (MARL), where multiple autonomous entities learn in the same environment. Multiple agents learning concurrently brings a set of challenges, such as partial observability, non-stationarity, and scalability; however, this setting is more realistic. Like in many real-life scenarios, the actions of one individual often affect the way others act and should therefore be considered in the learning process.

A sub-field of MARL called Cooperative Multi-Agent Reinforcement Learning (CMARL) focuses on learning policies for multiple agents that must coordinate their actions to achieve a shared objective. With a “divide and conquer” strategy, complex problems can thus be decomposed into smaller and simpler ones assigned to different agents acting in parallel in the environment. The problem of learning an optimal sequence of actions now also includes the need to strategically divide the task among agents that must learn

to coordinate and communicate in order to find the optimal way to accomplish the goal.

An emergent field of research in the RL community exploits finite-state machines to encode the structure of the reward function; ergo, the structure of the task at hand. This new construct, introduced by Toro Icarte et al. [19] and called *reward machine* (RM), can model non-Markovian reward and provides a symbolic interpretation of the stages required to complete a task. While the structure of the task is sometimes known in advance, it is rarely true in practice. Learning the RM has thus been the object of several works in recent years [3, 9, 10, 13, 21, 22] as a way to reduce the human input. In the multi-agent settings, however, the challenges highlighted before make the learning of the RM encoding the global task particularly difficult. Instead, it is more efficient to learn the RMs of smaller tasks obtained from an appropriate task decomposition. The “global” RM can later be reconstructed with a parallel composition of the RM of the sub-tasks.

Inspired by the work of Neary et al. [16] using RMs to decompose and distribute a global task to a team of collaborative agents, we propose a method combining task decomposition with autonomous learning of the RMs. Our approach offers a more scalable and interpretable way to learn the structure of a complex task involving collaboration among multiple agents. We present an algorithm that learns in parallel the RMs associated with the sub-task of each agent and their associated RL policies whilst guaranteeing that their executions achieve the global cooperative task. We experimentally show that with our method a team of agents is able to learn how to solve a collaborative task in two different environments.

## 2 BACKGROUND

This section gives a summary of the relevant background to make the paper self-contained. Given a finite set  $\mathcal{X}$ ,  $\Delta(\mathcal{X})$  denotes the probability simplex over  $\mathcal{X}$ ,  $\mathcal{X}^*$  denotes (possibly empty) sequences of elements from  $\mathcal{X}$ , and  $\mathcal{X}^+$  is a non-empty sequence. The symbols  $\perp$  and  $\top$  denote false and true, respectively.

### 2.1 Reinforcement Learning

We consider  $T$ -episodic labeled Markov decision processes (MDPs) [22], characterized by a tuple  $\langle \mathcal{S}, s_I, \mathcal{A}, p, r, T, \gamma, \mathcal{P}, l \rangle$  consisting of a set of states  $\mathcal{S}$ , an initial state  $s_I \in \mathcal{S}$ , a set of actions  $\mathcal{A}$ , a transition function  $p : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ , a not necessarily Markovian reward function  $r : (\mathcal{S} \times \mathcal{A})^+ \times \mathcal{S} \rightarrow \mathbb{R}$ , the time horizon  $T$  of each episode, a discount factor  $\gamma \in [0, 1]$ , a finite set of propositions  $\mathcal{P}$ , and a labeling function  $l : \mathcal{S} \times \mathcal{S} \rightarrow 2^\mathcal{P}$ .

A (state-action) history  $h_t = \langle s_0, a_0, \dots, s_t \rangle \in (\mathcal{S} \times \mathcal{A})^* \times \mathcal{S}$  is mapped into a *label trace* (or trace)  $\lambda_t = \langle l(s_0, s_1), \dots, l(s_{t-1}, s_t) \rangle \in (2^\mathcal{P})^+$  by applying the labeling function to each state transition in  $h_t$ . We assume that the reward function can be written in terms of a label trace, i.e. formally  $r(h_{t+1}) = r(\lambda_{t+1}, s_{t+1})$ . The goal is to find an optimal policy  $\pi : (2^\mathcal{P})^+ \times \mathcal{S} \rightarrow \mathcal{A}$ , mapping (traces-states) to

actions, in order to maximize the expected cumulative discounted reward  $R_t = \mathbb{E}_\pi \left[ \sum_{k=t}^T \gamma^{k-t} r(\lambda_{k+1}, s_{k+1}) \right]$ .

At time  $t$ , the agent keeps a trace  $\lambda_t \in (2^\mathcal{P})^+$ , and observes state  $s_t \in \mathcal{S}$ . The agent chooses the action to execute with its policy  $a = \pi(\lambda_t, s_t) \in \mathcal{A}$ , and the environment transitions to state  $s_{t+1} \sim p(\cdot | s_t, a_t)$ . As a result, the agent observes a new state  $s_{t+1}$  and a new label  $\mathcal{L}_{t+1} = l(s_t, s_{t+1})$ , and gets the reward  $r_{t+1} = r(\lambda_{t+1}, s_{t+1})$ . The trace  $\lambda_{t+1} = \lambda_t \oplus \mathcal{L}_{t+1}$  is updated with the new label.

In this work, we focus on the *goal-conditioned* RL problem [14] where the agent's task is to reach a goal state as rapidly as possible. We thus distinguish between two types of traces: *goal* traces and *incomplete* traces. A trace  $\lambda_t$  is said to be a *goal* trace if the task's goal has been achieved before  $t$  otherwise we say that the trace is *incomplete*. Formally, the MDP includes a *termination function*  $\tau : (2^\mathcal{P})^+ \times \mathcal{S} \rightarrow \{\perp, \top\}$  indicating that the trace  $\lambda_t$  at time  $t$  is an *incomplete* trace if  $\tau(\lambda_t, s_t) = \perp$  or a *goal* trace if  $\tau(\lambda_t, s_t) = \top$ . We assume the reward is 1 for goal traces and 0 otherwise. For ease of notation, we will refer to a goal trace as  $\lambda_t^\top$  and an incomplete trace as  $\lambda_t^\perp$ . Note that we assume the agent-environment interaction stops when the task's goal is achieved. This is without loss of generality as it is equivalent to having a final absorbing state with a null reward associated and no label returned by the labeling function  $l$ .

The single-agent framework above can be extended to the collaborative *multi-agent* setting as a Markov game. A cooperative *Markov game* of  $N$  agents is a tuple  $\mathcal{G} = \langle N, \mathcal{S}, s_I, \mathcal{A}, p, r, \tau, T, \gamma, \mathcal{P}, l \rangle$ , where  $\mathcal{S} = \mathcal{S}_1 \times \dots \times \mathcal{S}_N$  is the set of joint states,  $s_I \in \mathcal{S}$  is a joint initial state,  $\mathcal{A} = \mathcal{A}_1 \times \dots \times \mathcal{A}_N$  is the set of joint actions,  $p : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$  is a joint transition function,  $r : (\mathcal{S} \times \mathcal{A})^+ \times \mathcal{S} \rightarrow \mathbb{R}$  is the collective reward function,  $\tau : (2^\mathcal{P})^+ \times \mathcal{S} \rightarrow \{\perp, \top\}$  is the collective termination function, and  $l : \mathcal{S} \times \mathcal{S} \rightarrow 2^\mathcal{P}$  is an event labeling function.  $T$ ,  $\gamma$ , and  $\mathcal{P}$  are defined as for MDPs. Like Neary et al. [16], we assume each agent  $A_i$ 's dynamics are independently governed by local transition functions  $p_i : \mathcal{S}_i \times \mathcal{A}_i \rightarrow \Delta(\mathcal{S}_i)$ , hence the joint transition function is  $p(s' | s, a) = \prod_{i=1}^N p_i(s'_i | s_i, a_i)$  for all  $s, s' \in \mathcal{S}$  and  $a \in \mathcal{A}$ . The objective is to find a team policy  $\pi : (2^\mathcal{P})^+ \times \mathcal{S} \rightarrow \mathcal{A}$  mapping pairs of traces and joint states to joint actions that maximizes the expected cumulative collective reward.

*Example 2.1.* We use the THREEBUTTONS task [16], illustrated in Figure 1a, as a running example. The task consists of three agents ( $A_1$ ,  $A_2$ , and  $A_3$ ) that must cooperate for agent  $A_1$  to get to the *Goal* position. To achieve this, agents must open doors that prevent other agents from progressing by pushing buttons  $Y_B$ ,  $G_B$ , and  $R_B$ . The button of a given color opens the door of the same color and, two agents are required to push  $R_B$  at once hence involving synchronization. Once a door has been opened, it remains open until the end of the episode. The order in which high-level actions must be performed is shown in the figure: (1)  $A_1$  pushes  $Y_B$ , (2)  $A_2$  pushes  $G_B$ , (3)  $A_2$  and  $A_3$  push  $R_B$ , and (4)  $A_1$  reaches *Goal*. We can formulate this problem using the RL framework by defining a shared reward function returning 1 when the *Goal* position is reached. States solely consist of the position of the agents in the environment. The *proposition set* in this task is  $\mathcal{P} = \{R_B, Y_B, G_B, \neg R_B, A_2^R, A_2^{\neg R}, A_3^R, A_3^{\neg R}, Goal\}$ , where (i)  $R_B$ ,  $Y_B$  and  $G_B$  indicate that

the red, yellow and green buttons have been pushed respectively, (ii)  $A_i^R$  (resp.  $A_i^{\neg R}$ ), where  $i \in \{2, 3\}$ , indicates that agent  $A_i$  is (resp. has stopped) pushing the red button  $R_B$ , and (iii) *Goal* indicates that the goal position has been reached. A minimal *goal trace* is  $\langle \{\}, \{Y_B\}, \{G_B\}, \{A_2^R\}, \{A_3^R\}, \{R_B\}, \{Goal\} \rangle$ .

## 2.2 Reward Machines

We here introduce reward machines, the formalism we use to express decomposable (multi-agent) tasks.

### 2.2.1 Definitions.

A *reward machine* (RM) [19, 20] is a finite-state machine that represents the reward function of an RL task. Formally, an RM is a tuple  $M = \langle \mathcal{U}, \mathcal{P}, u_0, u_A, \delta_u, \delta_r \rangle$  where  $\mathcal{U}$  is a set of states;  $\mathcal{P}$  is a set of propositions;  $u_0 \in \mathcal{U}$  is the initial state;  $u_A \in \mathcal{U}$  is the final state;  $\delta_u : \mathcal{U} \times 2^\mathcal{P} \rightarrow \mathcal{U}$  is a state-transition function such that  $\delta_u(u, \mathcal{L})$  is the state that results from observing label  $\mathcal{L} \in 2^\mathcal{P}$  in state  $u \in \mathcal{U}$ ; and  $\delta_r : \mathcal{U} \times \mathcal{U} \rightarrow \mathbb{R}$  is a reward-transition function such that  $\delta_r(u, u')$  is the reward obtained for transitioning from state  $u \in \mathcal{U}$  to  $u' \in \mathcal{U}$ . We assume that (i) there are no outgoing transitions from  $u_A$ , and (ii)  $\delta_r(u, u') = 1$  if  $u' = u_A$  and 0 otherwise (following the reward assumption in Section 2.1). Given a trace  $\lambda = \langle \mathcal{L}_0, \dots, \mathcal{L}_n \rangle$ , a *traversal* (or run) is a sequence of RM states  $\langle v_0, \dots, v_{n+1} \rangle$  such that  $v_0 = u_0$ , and  $v_{i+1} = \delta_u(v_i, \mathcal{L}_i)$  for  $i \in [1, n]$ .

Ideally, RMs are such that (i) the cross product  $\mathcal{S} \times \mathcal{U}$  of their states with the MDP's make the reward and termination functions Markovian, and (ii) traversals for goal traces end in the final state  $u_A$  and traversals for incomplete traces do not.

*Example 2.2.* Figure 1b shows an RM for the THREEBUTTONS task. For simplicity, edges are labeled using the single proposition that triggers the transitions instead of sets. Note that the goal trace introduced in Example 2.1 ends in the final state  $u_A$ .

### 2.2.2 RL Algorithm.

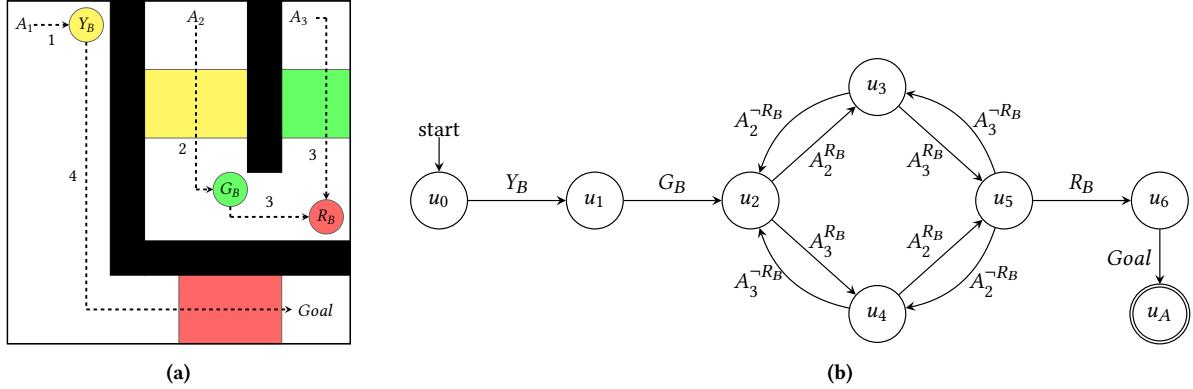
The Q-learning for RMs (QRM) algorithm [19, 20] exploits the task structure modeled through RMs. QRM learns a Q-function  $q_u$  for each state  $u \in \mathcal{U}$  in the RM. Given an experience tuple  $\langle s, a, s' \rangle$ , a Q-function  $q_u$  is updated as follows:

$$q_u(s, a) = q_u(s, a) + \alpha \left( \delta_r(u, u') + \gamma \max_{a'} q_{u'}(s', a') - q_u(s, a) \right),$$

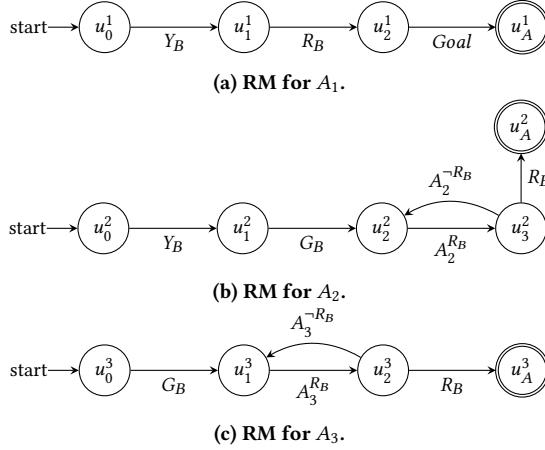
where  $u' = \delta_u(u, l(s, s'))$ . All Q-functions (or a subset of them) are updated at each step using the same experience tuple in a counterfactual manner. In the tabular case, QRM is guaranteed to converge to an optimal policy.

### 2.2.3 Multi-Agent Decomposition.

Neary et al. [16] recently proposed to decompose the RM for a multi-agent task into several RMs (one per agent) executed in parallel. The task decomposition into individual and independently learnable sub-tasks addresses the problem of non-stationarity inherent in the multi-agent setting. The use of RMs gives the high-level structure of the task each agent must solve. In this setting, each agent  $A_i$  has its own RM  $M_i$ , local state space  $\mathcal{S}_i$  (e.g., it solely observes its position in the grid), and propositions  $\mathcal{P}_i$  such that  $\mathcal{P} = \bigcup_i \mathcal{P}_i$ . Besides, instead of having a single opaque labeling function, each agent  $A_i$  employs its own labeling function  $l_i : \mathcal{S}_i \times \mathcal{S}_i \rightarrow 2^{\mathcal{P}_i}$ . Each labeling function is assumed to return at most one proposition per



**Figure 1: Illustration of the THREEBUTTONS grid (a) and a reward machine modeling the task’s structure (b) [16].**



**Figure 2: RMs for each of the agents in THREEBUTTONS [16].**

agent per timestep, and they should together output the same label as the global labeling function  $L$ .

Given a global RM  $M$  modeling the structure of the task at hand (e.g., that in Figure 1b) and each agent’s proposition set  $\mathcal{P}_i$ , Neary et al. [16] propose a method for deriving each agent’s RM  $M_i$  by projecting  $M$  onto  $\mathcal{P}_i$ . We refer the reader to Neary et al. [16] for a description of the projection mechanism and its guarantees since we focus on learning these individual RMs instead (see Section 3).

*Example 2.3.* Given the local proposition sets  $\mathcal{P}_1 = \{Y_B, R_B, Goal\}$ ,  $\mathcal{P}_2 = \{Y_B, G_B, A_2^{R_B}, A_2^{-R_B}, R_B\}$  and  $\mathcal{P}_3 = \{G_B, A_3^{R_B}, A_3^{-R_B}, R_B\}$ , Figure 2 shows the RMs that result from applying Neary et al. [16] projection algorithm.

Neary et al. [16] extend QRM and propose a decentralised training approach to train each agent in isolation (i.e., in the absence of their teammates) exploiting their respective RMs; crucially, the learnt policies should work when all agents interact simultaneously with the world. In both the individual and team settings, agents must synchronize whenever a shared proposition is observed (i.e., a proposition in the proposition set of two or more agents). Specifically, an agent should check with all its teammates whether their

labeling functions also returned the same proposition. In the individual setting, given that the rest of the team is not actually acting, synchronization is simulated with a fixed probability of occurrence.

### 3 LEARNING REWARD MACHINES IN COOPERATIVE MULTI-AGENT TASKS

Decomposing a task using RMs is a promising approach to solving collaborative problems where coordination and synchronization among agents are required, as described in Section 2.2.3. In well-understood problems, such as the THREEBUTTONS task, one can manually engineer the RM encoding the sequence of sub-tasks needed to achieve the goal. However, this becomes a challenge for more complex problems, where the structure of the task is unknown. In fact, the human-provided RM introduces a strong bias in the learning mechanism of the agents and can become adversarial if the human intuition about the structure of the task is incorrect. To alleviate this issue, we argue for learning each of the RMs automatically from traces collected via exploration instead of handcrafting them. In the following paragraphs, we describe two approaches to accomplish this objective.

#### 3.1 Learn a Global Reward Machine

A naive approach to learning each agent’s RM consists of two steps. First, a global RM  $M$  is learnt from traces where each label is the union of each agent’s label. Different approaches [9, 21, 22] could be applied in this situation. The learnt RM is then decomposed into one per agent by projecting it onto the local proposition set  $\mathcal{P}_i$  of each agent  $A_i$  [16].

Despite its simplicity, this method is prone to scale poorly. It has been observed that learning a minimal RM (i.e., an RM with the fewest states) from traces becomes significantly more difficult as the number of constituent states grows [9]. Indeed, the problem of learning a minimal finite-state machine from a set of examples is NP-complete [12]. Intuitively, the more agents interacting with the world, the larger the global RM will become, hence impacting performance and making the learning task more difficult. Besides, having multiple agents potentially increases the number of observable propositions as well as the length of the traces, hence increasing the complexity of the problem further.

## 3.2 Learn Individual Reward Machines

We propose to decompose the global task into sub-tasks, each of which can be independently solved by one of the agents. Note that even though an agent is assigned a sub-task, it does not have any information about its structure nor how to solve it. This is in fact what the proposed approach intends to tackle: learning the RM encoding the structure of the sub-task. This should be simpler than learning a global RM since the number of constituent states becomes smaller.

Given a set of  $N$  agents, we assume that the global task can be decomposed into  $N$  sub-task MDPs. Similarly to Neary et al. [16], we assume each agent’s MDP has its own state space  $\mathcal{S}_i$ , action space  $\mathcal{A}_i$ , transition function  $p_i$ , and termination function  $\tau_i$ . In addition, each agent has its own labeling function  $l_i$ ; hence, the agent may only observe a subset  $\mathcal{P}_i \subseteq \mathcal{P}$ . Note that the termination function is particular to each agent; for instance, agent  $A_2$  in THREEBUTTONS will complete its sub-task by pressing the green button, then the red button until it observes the signal  $R_B$  indicating that  $A_3$  is also pressing it. Ideally, the parallel composition of the learned RMs captures the collective goal.

We propose an algorithm that *interleaves* the induction of the RMs from a collection of label traces and the learning of the associated Q-functions, akin to that by Furelos-Blanco et al. [9] in the single agent setting. The decentralised learning of the Q-functions is performed using the method by Neary et al. [16], briefly outlined in Section 2.2.3. The induction of the RMs is done using a state-of-the-art inductive logic programming system called ILASP [15], which learns the transition function of an RM as a set of logic rules from example traces observed by the agent.

*Example 3.1.* The leftmost edge in Figure 2a corresponds to the rule  $\delta(u_0^1, u_1^1, T) :- \text{prop}(Y_B, T)$ . The  $\delta(X, Y, T)$  predicate expresses that the transition from  $X$  to  $Y$  is satisfied at time  $T$ , while the  $\text{prop}(P, T)$  predicate indicates that proposition  $P$  is observed at time  $T$ . The rule as a whole expresses that the transition from  $u_0^1$  to  $u_1^1$  is satisfied at time  $T$  if  $Y_B$  is observed at that time. The traces provided to RM learner are expressed as sets of prop facts; for instance, the trace  $\{\{Y_B\}, \{R_B\}, \{Goal\}\}$  is mapped into  $\{\text{prop}(Y_B, 0), \text{prop}(R_B, 1), \text{prop}(Goal, 2)\}$ .

Algorithm 1 shows the pseudocode describing how the reinforcement and RM learning processes are interleaved. The algorithm starts initializing for each agent  $A_i$ : the set of states  $\mathcal{U}^i$  of the RM, the RM  $M^i$  itself, the associated Q-functions  $q^i$ , and the sets of example goal ( $\Lambda_T^i$ ) and incomplete ( $\Lambda_\perp^i$ ) traces (1.1–5). Note that, initially, each agent’s RM solely consists of two unconnected states: the initial and final states (i.e., the agent loops in the initial state forever). Next,  $NumEpisodes$  are run for each agent. Before running any steps, each environment is reset to the initial state  $s_I^i$ , each RM  $M^i$  is reset to its initial state  $u_0^i$ , each agent’s trace  $\lambda^i$  is empty, and we indicate that no agents have yet completed their episodes (1.8–9). Then, while there are agents that have not yet completed their episodes (1.10), a training step is performed for each of the agents, which we describe in the following paragraphs. Note that we show a sequential implementation of the approach but it could be performed in parallel to improve performance.

**Algorithm 1:** Multi-Agent QRM with RM Learning

```

1 for  $i = 1$  to  $N$  do
2    $\mathcal{U}^i \leftarrow \{u_0^i, u_A^i\}$ 
3    $M^i \leftarrow \text{InitializeRM}(\mathcal{U}^i)$ 
4    $q^i \leftarrow \text{InitializeQ}(M^i)$ 
5    $\Lambda_T^i \leftarrow \emptyset, \Lambda_\perp^i \leftarrow \emptyset$ 
6 for  $n = 1$  to  $NumEpisodes$  do
7    $t \leftarrow 0$ 
8   for  $i = 1$  to  $N$  do
9      $u_t^i \leftarrow u_0^i, s_t^i \leftarrow s_I^i, \lambda_t^i \leftarrow \emptyset, done^i \leftarrow \perp$ 
10    while  $\exists j \in N$  such that  $done^j = \perp$  do
11      for  $i = 1$  to  $N$  do
12        TrainAgent( $done^i, s_t^i, u_t^i, q^i, \lambda_t^i, \Lambda_G^i, \Lambda_I^i, M^i, \mathcal{U}^i$ )
13       $t \leftarrow t + 1$ 
14
15 Procedure TrainAgent( $done, s_t, u_t, q, \lambda_t, \Lambda_T, \Lambda_\perp, M, \mathcal{U}$ )
16   if  $done = \perp$  then
17      $a \leftarrow \text{GetAction}(q_{u_t}, s_t)$ 
18      $s_{t+1}, isGoalAchieved \leftarrow \text{EnvStep}(s_t, a)$ 
19      $\mathcal{L}_{t+1} \leftarrow l(s_t, s_{t+1})$ 
20      $\lambda_{t+1} \leftarrow \lambda_t \oplus \mathcal{L}_{t+1}$ 
21      $r, u_{t+1} \leftarrow \text{GetNextRMState}(M, u_t, \mathcal{L}_{t+1})$ 
22      $q_{u_t} \leftarrow \text{UpdateQ}(q_{u_t}, s_t, a, r, s_{t+1}, q_{u_{t+1}})$ 
23     for  $u \in \mathcal{U} \setminus \{u_t\}$  do
24        $r, u' \leftarrow \text{GetNextRMState}(M, u, \mathcal{L}_{t+1})$ 
25        $q_u \leftarrow \text{UpdateQ}(q_u, s_t, a, r, s_{t+1}, q_{u'})$ 
26
27   if  $isGoalAchieved$  then
28      $\Lambda_T \leftarrow \Lambda_T \cup \{\lambda_{t+1}\}$ 
29      $\Lambda_\perp \leftarrow \Lambda_\perp \cup \text{GenerateIncompleteTraces}(\lambda_{t+1})$ 
30      $M_{new} \leftarrow \text{LearnRM}(\mathcal{U}, \Lambda_T, \Lambda_\perp)$ 
31      $done \leftarrow \top$ 
32   else if  $u_{t+1} = u_A$  then
33      $\Lambda_\perp \leftarrow \Lambda_\perp \cup \{\lambda_{t+1}\}$ 
34      $M_{new} \leftarrow \text{LearnRM}(\mathcal{U}, \Lambda_T, \Lambda_\perp)$ 
35      $done \leftarrow \top$ 
36   else if  $t + 1 = T$  then
37      $\Lambda_\perp \leftarrow \Lambda_\perp \cup \{\lambda_{t+1}\}$ 
38      $done \leftarrow \top$ 
39
40   if  $M \neq M_{new}$  then
41      $M \leftarrow M_{new}$ 
42      $q \leftarrow \text{InitializeQ}(M)$ 
43
44 Procedure LearnRM( $\mathcal{U}, \Lambda_T, \Lambda_\perp$ )
45    $M \leftarrow \text{RM Learner}(\mathcal{U}, \Lambda_T, \Lambda_\perp)$ 
46   while  $M = \perp$  do
47      $\mathcal{U} \leftarrow \mathcal{U} \cup \{u_{|\mathcal{U}|-1}\}$ 
48      $M \leftarrow \text{RM Learner}(\mathcal{U}, \Lambda_T, \Lambda_\perp)$ 
49   return  $M$ 

```

The TrainAgent routine shows how the reinforcement learning and RM learning processes are interleaved for a given agent  $A_i$  operating in its own environment. From 1.17 to 1.25, the steps for QRM (i.e., the RL steps) are performed. The agent first selects the next action  $a$  to execute in its current state  $s_t$  given the Q-function  $q_{u_t}$  associated with the current RM state  $u_t$ . The action is then applied, and the agent observes the next state  $s_{t+1}$  and whether it has achieved its goal (1.18). The next label  $\mathcal{L}_{t+1}$  is then determined (1.19) and used to (i) extend the episode trace  $\lambda_t$  (1.20), and (ii) obtain reward  $r$  and the next RM state  $u_{t+1}$  (1.21). The Q-function  $q$  is updated for both the RM state  $u_t$  the agent is in at time  $t$  (1.22) and in a counterfactual manner for all the other states of the RM (1.25).

The learning of the RMs occurs from 1.27 to 1.42. Remember that there are two sets of example traces for each agent: one for goal traces  $\Lambda_\top$  and one for incomplete traces  $\Lambda_\perp$ . Crucially, the learnt RMs must be such that (i) traversals for goal traces end in the final state, and (ii) traversals for incomplete traces do not end in the final state. Given the trace  $\lambda_{t+1}$  and the RM state  $u_{t+1}$  at time  $t+1$ , the example sets are updated in three cases:

- (1) If  $\lambda_{t+1}$  is a goal trace (1.27),  $\lambda_{t+1}$  is added to  $\Lambda_\top$ . We consider as incomplete all sub-traces of  $\lambda_\top$ :  $\{\langle \mathcal{L}_0, \dots, \mathcal{L}_k \rangle; \forall k \in [0, \dots, |\lambda_\top| - 1]\}$  (see Example 3.2). Note that if a sub-trace was a goal trace, the episode would have ended before. This optimization enables capturing incomplete traces faster; that is, it prevents waiting for them to appear as counterexamples (see next case).
- (2) If  $\lambda_{t+1}$  is an incomplete trace and  $u_{t+1}$  is the final state of the RM (1.32), then  $\lambda_{t+1}$  is a counterexample since reaching the final state of the RM should be associated with completing the task. In this case, we add  $\lambda_{t+1}$  to  $\Lambda_\perp$ .
- (3) If  $\lambda_{t+1}$  is an incomplete trace and the maximum number of steps per episode  $T$  has been reached (1.36), we add  $\lambda_{t+1}$  to  $\Lambda_\perp$ .

In the first two cases, a new RM is learnt once the example sets have been updated. The LearnRM routine (1.44–49) finds the RM with the fewest states that covers the example traces. The RM learner (here ILASP) is initially called using the current set of RM states  $\mathcal{U}$  (1.45); then, if the RM learner finds no solution covering the provided set of examples, the number of states is increased by 1. This approach guarantees that the RMs learnt at the end of the process are *minimal* (i.e., consist of the fewest possible states) [9]. Finally, like Furelos-Blanco et al. [9], we reset the Q-functions associated with an RM when it is relearned; importantly, the Q-functions depend on the RM structure and, hence, they are not easy to keep when the RM changes (1.42). In all three enumerated cases, the episode is interrupted.

*Example 3.2.* Given the sub-task assigned to the agent  $A_3$  in the THREEBUTTONS environment described in Figure 2c, we consider a valid goal trace  $\lambda = \langle \{\}, \{G_B\}, \{A_3^{R_B}\}, \{A_3^{\neg R_B}\}, \{A_3^{R_B}\}, \{R_B\} \rangle$ . The set of generated incomplete traces is:

$$\begin{aligned} &\langle \{\}, \\ &\langle \{\}, \{G_B\}, \\ &\langle \{\}, \{G_B\}, \{A_3^{R_B}\}, \\ &\langle \{\}, \{G_B\}, \{A_3^{R_B}\}, \{A_3^{\neg R_B}\}, \end{aligned}$$

$$\langle \{\}, \{G_B\}, \{A_3^{R_B}\}, \{A_3^{\neg R_B}\}, \{A_3^{R_B}\} \rangle$$

We have described in this section a method to learn the RM associated with each sub-task. In the next section, we evaluate how this approach performs on two collaborative tasks.

## 4 EXPERIMENTS

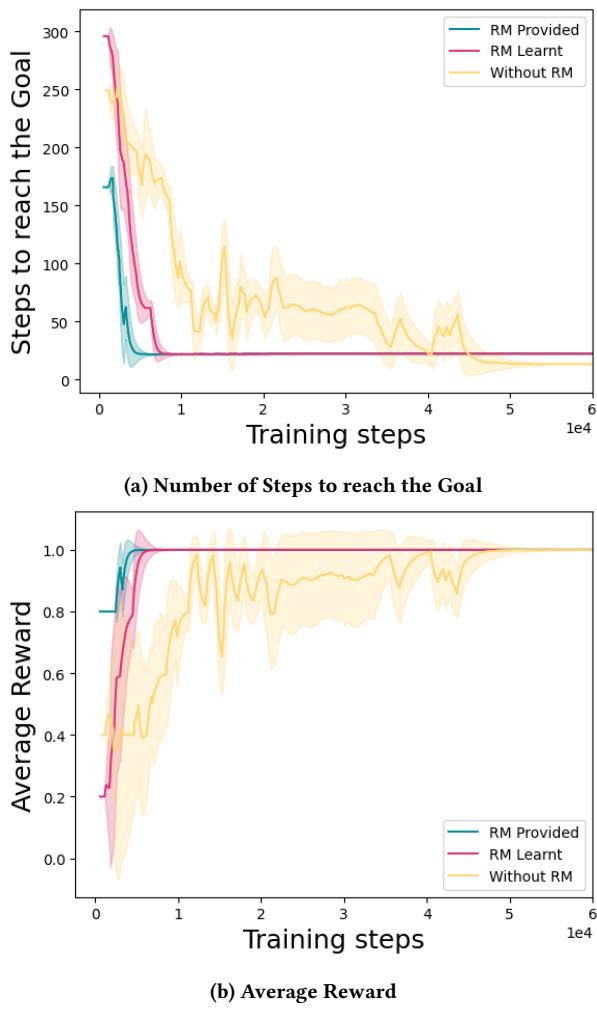
We evaluate our approach by looking at two metrics as the training progresses: (1) the collective reward obtained by the team of agents and (2) the number of steps required for the team to achieve the goal. The first metric indicates whether the team learns how to solve the task by getting a reward of 1 upon completion. The second one estimates the quality of the solution, i.e. the lower the number of steps the more efficient the agents. Although the training is performed in isolation for each agent, the results presented here were evaluated with all the agents acting together in a shared environment. Additionally, we inspect the RM learnt by each agent and perform a qualitative assessment of their quality. The results presented show the mean value and the 95%-confidence interval of the metrics evaluated over 5 different random seeds in two grid-based environments of size 7x7. We ran the experiments on an EC2 C5.2XLARGE instance on AWS using Python 3.7.0. We use the state-of-the-art inductive logic programming system ILASP v4.2.0 [15] to learn the RM from the set of traces with a timeout set to 1h.

### 4.1 THREEBUTTONS Task

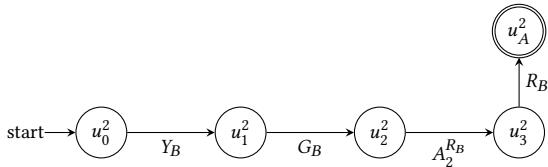
We evaluate our approach in the THREEBUTTONS task described in Example 2.1. The number of steps required to complete the task and the collective reward obtained by the team of agents throughout training are shown in Figure 3. The *green* curve shows the performance of the team of agents when the local RMs are manually engineered and provided. In *pink*, we show the performance of the team when the RMs are learnt. Finally in *yellow*, we show the performance of the team when each agent learns independently to perform their task without using the RM framework<sup>1</sup>. We observe that regardless of whether the local RMs are handcrafted or learned, the agents learn policies that, when combined, lead to the completion of the global task. Indeed, the collective reward obtained converges to the maximum reward indicating that all the agents have learnt to execute their sub-tasks and coordinate in order to achieve the common goal. Learning the RMs also helps speed up the learning for each of the agent, significantly reducing the number of training iterations required to learn to achieve the global task. We note however that the policy learnt without the RM framework requires less steps than when the RMs are used.

We present in Figure 4 the RM learnt by  $A_2$  using our interleaved learning approach. We compare it to the “true” RM shown in Figure 2b. The back transition from  $u_3^2$  to  $u_2^2$  associated with the label  $A_2^{\neg R_B}$  is missing from the learnt RM. This can be explained by the fact that this transition does not contribute towards the goal. When deriving the minimal RM from the set of labels, this one in particular is not deemed important and is thus ignored by the RM learner.

<sup>1</sup>This is equivalent of using a 2-State RM, reaching the final state only when the task is completed.



**Figure 3: Comparison between handcrafted RMs (RM Provided) and our approach learning the RMs from traces (RM Learnt) in the THREEBUTTONS environment.**



**Figure 4: Learnt RM for  $A_2$ .**

Learning the global RM was also attempted in this environment. Unfortunately, learning the minimal RM for the size of this problem is hard and the RM learner timed out after the 1h limit.

## 4.2 RENDEZVOUS Task

The second environment in which we evaluate our approach is the 2-agent RENDEZVOUS task [16] presented in Figure 5a. This task is composed of 2 agents acting in a grid with the ability to go UP,

DOWN, LEFT, RIGHT or DO NOTHING. The task requires the agents to move in the environment to first meet in an RDV location, i.e, all agents need to simultaneously be in that location for at least one timestep. Then each of the agents must reach their individual goal location for the global task to be completed.

We present in Figure 6 the number of steps (6a) to complete the task and the collective reward received by the team of agents (6b) throughout training. In green we show the performance when the RMs are known a priori and provided to the agents. This scenario, where we have perfect knowledge about the structure of each sub-task, is used as an upper bound for the results of our approach. In pink, we show the performances of our approach where the local RMs are learnt. The yellow curve shows the performance of the team when each agent learns to perform their individual task without the RM construct.

The collective reward obtained while interleaving the learning of the policies and the learning of the local RMs converges to the maximum reward after a few iterations. In this task, the RM framework is shown to be crucial to help the team solve the task in a timely manner. Indeed, without the RMs (either known a priori or learnt), the team of agents do not succeed at solving the task.

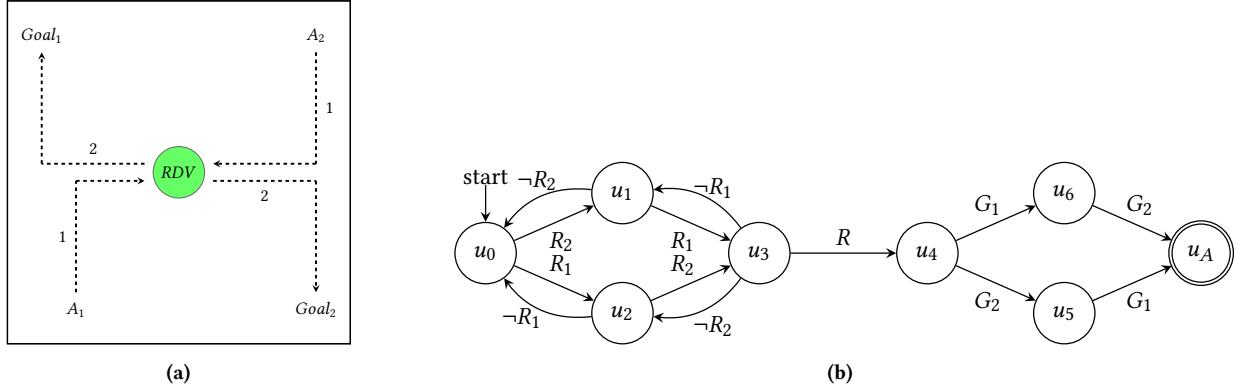
We also attempted to learn the global RM directly but the task could not be solved (i.e., a collective goal trace was not observed). The collective reward remained null throughout the training that we manually stopped after  $1e6$  timesteps.

## 5 RELATED WORK

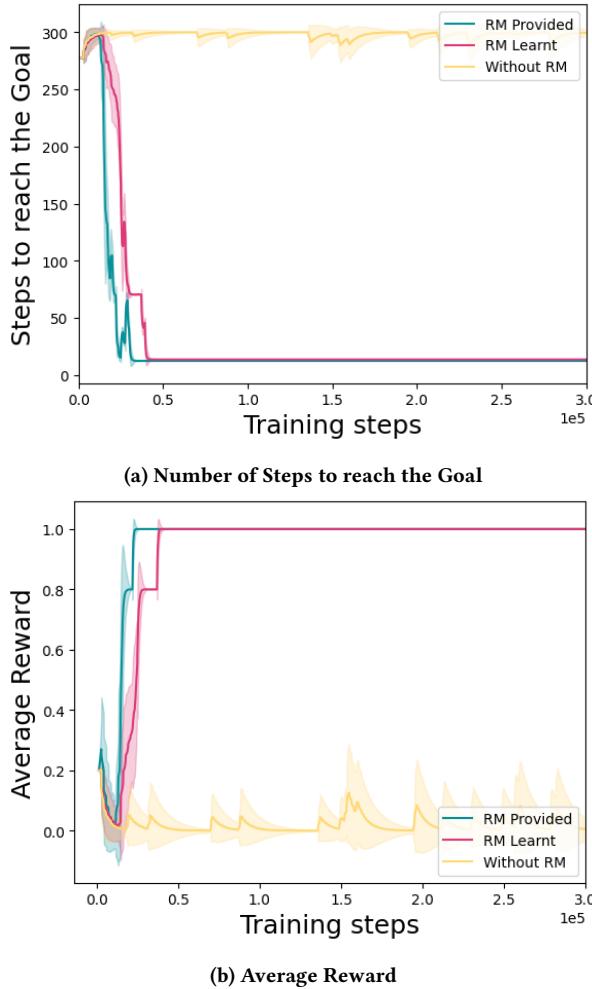
Since the recent introduction of reward machines as a way to model non-Markovian rewards [19, 20], several lines of research have based their work on this concept (or similar finite-state machines). Most of the work has focused on how to derive them from logic specifications [1] or demonstrations [2], as well as learning them using different methods [3, 9–11, 13, 21, 22].

In this work, based on the multi-agent with RMs work by Neary et al. [16], the RMs of the different agents are executed in parallel. Recent works have considered other ways of composing RMs, such as merging the state and reward transition functions [6] or arranging them hierarchically by enabling RMs to call each other [10]. The latter is a natural way of extending this work since agents could share subroutines in the form of callable RMs.

The policy learning algorithm for exploiting RMs we employ is an extension of the QRM algorithm (see Sections 2.2.2–2.2.3). However, algorithms based on making decisions at two hierarchical levels [9, 19] or more [10] have been proposed. In the simplest case (i.e., two levels), the agent first decides which transition to satisfy from a given RM state, and then decides which low-level actions to take to satisfy that transition. For example, if agent  $A_2$  gets to state  $u_3^2$  in Figure 2b, the decisions are: (1) whether to satisfy the transition labeled  $A_2^{R_B}$  or the one labeled  $R_B$ , and (2) given the chosen transition, act towards satisfying it. Unlike QRM, this method is not guaranteed to learn optimal policies; however, it enables lower-level sub-tasks to be reused while QRM does not. Note that the Q-functions learned by QRM depend on the structure of the whole RM and, hence, we learn them from scratch every time a new RM is induced. While other methods attempt to leverage the previously learned Q-functions [22], the hierarchical approach



**Figure 5: Example of the RENDEZVOUS task where 2 agents must meet on the RDV point (green) before reaching their goal state  $G_1$  and  $G_2$  for agents  $A_1$  and  $A_2$  respectively.**



**Figure 6: Comparison between handcrafted RMs (RM Provided) and our approach learning the RMs from traces (RM Learnt) in the RENDEZVOUS environment.**

outlined here does not need to relearn all functions each time a new RM is induced.

In the multi-agent community, the use of finite-state machines for task decomposition in collaborative settings has been studied in [4, 8] where a top-down approach is adopted to construct sub-machines from a global task. Unlike our work, these methods focus on the decomposition of the task but not on how to execute it. The algorithm presented in this paper interleaves both the decomposition and the learning of the policies associated with each state of the RM.

The MARL community studying the collaborative setting has proposed different approaches to decompose the task among all the different agents. For example, QMIX [17] tackles the credit assignment problem in MARL assuming that the team's Q-function can be factorized, and performs a value-iteration method to centrally train policies that can be executed in a decentralised fashion. While this approach has shown impressive empirical results, its interpretation is more difficult than understanding the structure of the task using RM.

The use of RMs in the MARL literature is starting to emerge. The work of Neary et al. [16] was the first to propose the use of RMs for task decomposition among multiple agents. As already mentioned in this paper, this work assumes that the structure of the task is known 'a priori', which is often untrue. With a different purpose, Dann et al. [5] propose to use RMs to help an agent predict the next actions the other agents in the system will perform. Instead of modeling the other agents' plan (or program), the authors argue that RMs provide a more permissive way to accommodate for variations in the behaviour of the other agent by providing a higher-level mechanism to specify the structure of the task. In this work as well, the RMs are pre-defined and provided to the agent.

Finally, the work of Eappen and Jagannathan [7] uses a finite-state machine called 'task monitor', which is built from temporal logic specifications that encode the reward signal of the agents in the system. A task monitor is akin to an RM with some subtle differences like the use of registers for memory.

## 6 CONCLUSION AND FUTURE WORK

Dividing a problem into smaller parts that are easier to solve is a natural approach performed instinctively by humans. It is also particularly suited for the multi-agent setting where all the available resources can be utilized to solve the problem or when a complementary set of skills is required.

In this work, we extend the approach by Neary et al. [16], who leverage RMs for task decomposition in a multi-agent setting. We propose a novel approach where the RMs are learnt instead of manually engineered. The method presented in this paper interleaves the learning of the RM from the traces collected by the agent and the learning of the set of policies used to achieve the task’s goal. Learning the RMs associated with each sub-task not only helps deal with non-Markovian rewards but also provides a more interpretable way to understand the structure of the task. We show experimentally in the THREEBUTTONS and the RENDEZVOUS environments that our approach converges to the maximum reward a team of agents can obtain by collaborating to achieve a goal. Finally, we validated that the learnt RM corresponds indeed to the RM needed to achieve each sub-task.

While our approach lifts the assumption about knowing the structure of the task ‘a priori’, a set of challenges remain and could be the object of further work:

- (1) A labeling function is assumed to exist for each of the agents and be known by them. In other words, each agent knows the labels relevant to its task and any labels used for synchronization with the other team members.
- (2) If the labeling functions are noisy, agents may fail to synchronize and hence not be able to progress in their sub-tasks. In this scenario, the completion of the cooperative task is compromised. Defining or learning a communication protocol among the agents could be the object of future work.
- (3) Our approach assumes that the global task has been automatically divided into several tasks, each to be completed by one of the agents. Dynamically creating these sub-tasks and assigning them to each of the agents is crucial to having more autonomous agents and would make an interesting topic for further research.
- (4) We leverage task decomposition to learn simpler RMs that when run in parallel represent a more complex global RM. Task decomposition can be driven further through task hierarchies, which could be represented through hierarchies of RMs [10]. Intuitively, each RM in the hierarchy should be simpler to learn and enable reusability and further parallelization among the different agents.

The use of RMs in the multi-agent context is still in its infancy and there are several exciting avenues for future work. We have demonstrated in this paper that it is possible to learn them under certain assumptions that could be relaxed in future work.

## ACKNOWLEDGMENTS

Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-22-2-0243. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government

is authorised to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

## REFERENCES

- [1] 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 *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 6065–6073.
- [2] Alberto Camacho, Jacob Varley, Andy Zeng, Deepali Jain, Atil Iscen, and Dmitry Kalashnikov. 2021. Reward Machines for Vision-Based Robotic Manipulation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 14284–14290.
- [3] Phillip J. K. Christoffersen, Andrew C. Li, Rodrigo Toro Icarte, and Sheila A. McIlraith. 2020. Learning Symbolic Representations for Reinforcement Learning of Non-Markovian Behavior. In *Proceedings of the Knowledge Representation and Reasoning Meets Machine Learning (KR2ML) Workshop at the Advances in Neural Information Processing Systems (NeurIPS) Conference*.
- [4] Jin Dai and Hai Lin. 2014. Automatic synthesis of cooperative multi-agent systems. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*. 6173–6178.
- [5] Michael Dann, Yuan Yao, Natasha Alechina, Brian Logan, and John Thangarajah. 2022. Multi-Agent Intention Progression with Reward Machines. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 215–222.
- [6] Giuseppe De Giacomo, Marco Favorito, Luca Iocchi, Fabio Patrizi, and Alessandro Ronca. 2020. Temporal Logic Monitoring Rewards via Transducers. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*. 860–870.
- [7] Joe Eappen and Suresh Jagannathan. 2022. DistSPECTRL: Distributing Specifications in Multi-Agent Reinforcement Learning Systems. *arXiv preprint arXiv:2206.13754* (2022).
- [8] Ayman Elshenawy Elsefy. 2020. A task decomposition using (HDec-POSMDPs) approach for multi-robot exploration and fire searching. *International Journal of Robotics and Mechatronics* 7, 1 (2020), 22–30.
- [9] Daniel Furelos-Blanco, Mark Law, Anders Jonsson, Krysia Broda, and Alessandra Russo. 2021. Induction and Exploitation of Subgoal Automata for Reinforcement Learning. *J. Artif. Intell. Res.* 70 (2021), 1031–1116.
- [10] Daniel Furelos-Blanco, Mark Law, Anders Jonsson, Krysia Broda, and Alessandra Russo. 2022. Hierarchies of Reward Machines. *arXiv preprint arXiv:2205.15752* (2022).
- [11] Maor Gaon and Ronen I. Brafman. 2020. Reinforcement Learning with Non-Markovian Rewards. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 3980–3987.
- [12] E. Mark Gold. 1978. Complexity of Automaton Identification from Given Data. *Inf. Control.* 37, 3 (1978), 302–320.
- [13] Mohammadhosein Hasanbeig, Natasha Yogananda Jeppu, Alessandro Abate, Tom Melham, and Daniel Kroening. 2021. DeepSynth: Automata Synthesis for Automatic Task Segmentation in Deep Reinforcement Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*. 7647–7656.
- [14] Leslie Pack Kaelbling. 1993. Learning to Achieve Goals. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 1094–1099.
- [15] Mark Law, Alessandra Russo, and Krysia Broda. 2015. The ILASP System for Learning Answer Set Programs. <https://www.doc.ic.ac.uk/~ml1909/ILASP>
- [16] Cyrus Neary, Zhe Xu, Bo Wu, and Ufuk Topcu. 2021. Reward Machines for Cooperative Multi-Agent Reinforcement Learning. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 934–942.
- [17] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2020. Monotonic value function factorisation for deep multi-agent reinforcement learning. *The Journal of Machine Learning Research* 21, 1 (2020), 7234–7284.
- [18] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. MIT Press.
- [19] 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 *Proceedings of the International Conference on Machine Learning (ICML)*. 2107–2116.
- [20] Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. 2022. Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning. *J. Artif. Intell. Res.* 73 (2022), 173–208.
- [21] Rodrigo Toro Icarte, Ethan Waldie, Toryn Q. Klassen, Richard Anthony Valenzano, Margarita P. Castro, and Sheila A. McIlraith. 2019. Learning Reward Machines for Partially Observable Reinforcement Learning. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS) Conference*. 15497–15508.
- [22] Zhe Xu, Ivan Gavran, Yousef Ahmad, Rupak Majumdar, Daniel Neider, Ufuk Topcu, and Bo Wu. 2020. Joint Inference of Reward Machines and Policies for Reinforcement Learning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. 590–598.