Reward Machines for Cooperative Multi-Agent Reinforcement Learning

Cyrus Neary
The University of Texas at Austin
cneary@utexas.edu

Zhe Xu
Arizona State University
xzhe1@asu.edu

Bo Wu
The University of Texas at Austin
bowu86@gmail.com

Ufuk Topcu
The University of Texas at Austin
utopcu@utexas.edu

ABSTRACT

In cooperative multi-agent reinforcement learning, a collection of agents learns to interact in a shared environment to achieve a common goal. We propose the use of reward machines (RM) — Mealy machines used as structured representations of reward functions — to encode the team's task. The proposed novel interpretation of RMs in the multi-agent setting explicitly encodes required teammate interdependencies, allowing the team-level task to be decomposed into sub-tasks for individual agents. We define such a notion of RM decomposition and present algorithmically verifiable conditions guaranteeing that distributed completion of the sub-tasks leads to team behavior accomplishing the original task. This framework for task decomposition provides a natural approach to decentralized learning: agents may learn to accomplish their sub-tasks while observing only their local state and abstracted representations of their teammates. We accordingly propose a decentralized q-learning algorithm. Furthermore, in the case of undiscounted rewards, we use local value functions to derive lower and upper bounds for the global value function corresponding to the team task. Experimental results in three discrete settings exemplify the effectiveness of the proposed RM decomposition approach, which converges to a successful team policy an order of magnitude faster than a centralized learner and significantly outperforms hierarchical and independent q-learning approaches.

KEYWORDS

Decentralized Multi-Agent Learning; Discrete Event Systems; Task Decomposition; Bisimulation

ACM Reference Format:
Cyrus Neary, Zhe Xu, Bo Wu, and Ufuk Topcu. 2021. Reward Machines for Cooperative Multi-Agent Reinforcement Learning. In *Proc.of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), Online, May 3–7, 2021*, IFAAMAS, 9 pages.

1 INTRODUCTION

In multi-agent reinforcement learning (MARL), a collection of agents learn to maximize expected long-term return through interactions with each other and with a shared environment. We study MARL in a cooperative setting: all of the agents are rewarded collectively for achieving a team task.

Proc.of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), U. Endriss, A. Nowé, F. Dignum, A. Lomuscio (eds.), May 3–7, 2021, Online.

Two challenges inherent to MARL are coordination and non-stationarity. Firstly, coordination is needed between agents because the correctness of any individual agent's actions may depend on the actions of its teammates [3, 10]. Secondly, the agents are learning and updating their behaviors simultaneously. From the point of view of any individual agent, the learning problem is non-stationary; the best solution for any individual is constantly changing [11].

A reward machine (RM) is a Mealy machine used to define tasks and behaviors dependent on abstracted descriptions of the environment [14]. Intuitively, RMs allow agents to separate tasks into stages and to learn different sets of behaviors for the different portions of the overall task. In this work, we use RMs to describe cooperative tasks and we introduce a notion of RM decomposition for the MARL problem. The proposed use of RMs explicitly encodes the information available to each agent, as well as the teammate communications necessary for successful cooperative behavior. The global (cooperative) task can then be decomposed into a collection of new RMs, each encoding a sub-task for an individual agent. We propose a decentralized learning algorithm that trains the agents individually using these sub-task RMs, effectively reducing the team's task to a collection of single-agent reinforcement learning problems. The algorithm assumes each agent may only observe its own local state and the information encoded in its sub-task RM.

Furthermore, we provide conditions guaranteeing that if each agent accomplishes its sub-task, the corresponding joint behavior accomplishes the team task. Finally, decomposition of the team's task allows for each agent to be trained independently of its teammates and thus addresses the problems posed by non-stationarity. Individual agents condition their actions on abstractions of their teammates, eliminating the need for simultaneous learning.

Experimental results in three discrete domains exemplify the strengths of the proposed decentralized algorithm. In a two-agent rendezvous task, the proposed algorithm converges to successful team behavior more than an order of magnitude faster than a centralized learner and performs roughly on par with hierarchical independent learners (h-IL) [35]. In a ten-agent variant of the task, the proposed algorithm quickly learns effective team behavior while neither h-IL nor independent q-learning (IQL) [34] converges to policies completing the task within the allowed training steps.

2 PRELIMINARIES

A Markov decision process (MDP) is a tuple $\mathcal{M} = \langle S, A, r, p, \gamma \rangle$ consisting of a finite set of states S, a finite set of actions A, a reward function $r : S \times A \times S \to \mathbb{R}$, a transition probability function

$p : S \times A \to \Delta(S)$, and a discount factor $\gamma \in (0, 1]$. Here $\Delta(S)$ is the set of all probability distributions over S. We denote by $p(s'|s, a)$ the probability of transitioning to state s' from state s under action a. A stationary policy $\pi : S \to \Delta(A)$ maps states to probability distributions over the set of actions. In particular, if an agent is in state $s_t \in S$ at time step t and is following policy π, then $\pi(a_t|s_t)$ denotes its probability of taking action $a_t \in A$.

The goal of reinforcement learning (RL) is to learn an optimal policy π^* maximizing the expected sum of discounted future rewards from any state [33]. The q-function for policy π is defined as the expected discounted future reward that results from taking action a from state s and following policy π thereafter. Tabular q-learning [39], an RL algorithm, uses the experience $\{(s_t, a_t, r_t, s_{t+1})\}_{t\in\mathbb{N}_0}$ of an agent interacting with an MDP to learn the q-function $q^*(s, a)$ corresponding to an optimal policy π^*. Given $q^*(s, a)$, the optimal policy may be recovered.

A common framework used to extend RL to a multi-agent setting is the Markov game [3, 23]. A cooperative Markov game of N agents is a tuple $\mathcal{G} = \langle S_1, ..., S_N, A_1, ..., A_N, p, R, \gamma\rangle$. S_i and A_i are the finite sets of agent i's local states and actions respectively. We define the set of joint states as $\mathcal{S} = S_1 \times ... \times S_N$ and we similarly define the set of joint actions to be $\mathcal{A} = A_1 \times ... \times A_N$. $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is a joint state transition probability distribution. $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the team's collective reward function which is shared by all agents, and $\gamma \in (0, 1]$ is a discount factor.

In this work, we assume the dynamics of each agent are independently governed by local transition probability functions $p_i : S_i \times A_i \to \Delta(S_i)$. The joint transition function is then constructed as $p(\mathbf{s}'|\mathbf{s}, \mathbf{a}) = \Pi_{i=1}^{N} p_i(s_i'|s_i, a_i)$, for all $\mathbf{s}, \mathbf{s}' \in \mathcal{S}$ and $\mathbf{a} \in \mathcal{A}$. A team policy is defined as $\boldsymbol{\pi} : \mathcal{S} \to \Delta(\mathcal{A})$. As in the single agent case, the objective of team MARL is to find a team policy $\boldsymbol{\pi}^*$ maximizing expected discounted future reward from any joint state.

3 REWARD MACHINES FOR MARL

To introduce reward machines (RM) and to illustrate how they may be used to encode a team's task, we consider the example shown in Figure 1a. Three agents, denoted A_1, A_2, and A_3, operate in a shared environment with the objective of allowing A_1 to reach the goal location denoted *Goal*. However, the red, yellow, and green colored regions are blocking the paths of agents A_1, A_2, and A_3 respectively. To allow the agents to cross these colored regions, the button of the corresponding color must be pressed first. Furthermore, the yellow and green buttons may be pressed by an individual agent, but the red button requires two agents to simultaneously occupy the button's location before it is activated. The dashed and numbered arrows in the figure illustrate the sequence of events necessary for task completion: A_1 should push the yellow button allowing A_2 to proceed to the green button, which is necessary for A_3 to join A_2 in pressing the red button, finally allowing A_1 to cross the red region and reach *Goal*.

3.1 Reward Machines for Task Description

Definition 1. *A* reward machine (RM) $\mathcal{R} = \langle U, u_I, \Sigma, \delta, \sigma\rangle$ *consists of a finite, nonempty set* U *of states, an initial state* $u_I \in U$*, a finite set* Σ *of environment events, a transition function* $\delta : U\times\Sigma \to U$*, and an output function* $\sigma : U \times U \to \mathbb{R}$.

Reward machines are a type of Mealy machine used to define temporally extended tasks and behaviors. Initially introduced in [14], we adapt the RM's definition slightly to better suit the multi-agent setting. In particular, we define the transition function to take events $e \in \Sigma$ as input, instead of collections of atomic propositions. This change simplifies the notation required for task decomposition and it is relatively minor; any collection of atomic propositions can be redefined as an individual event e. Furthermore, the output function σ is defined to map transitions to constant values, instead of to reward functions. Finally, we note that δ and σ are partial functions; they are defined on subsets of $U\times\Sigma$ and $U\times U$ respectively. Figure 1b illustrates the RM encoding the buttons task.

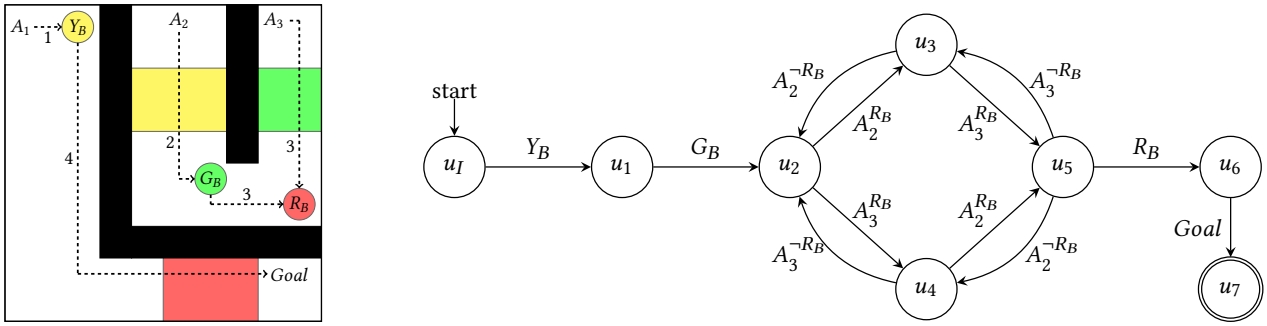
Set Σ is the collection of all the high-level events necessary to describe the team's task. For example, the event set corresponding to the buttons task from Figure 1a is $\Sigma = \{Y_B, G_B, A_2^{R_B}, A_2^{\neg R_B}, A_3^{R_B}, A_3^{\neg R_B}, R_B, Goal\}$. Here, events Y_B, G_B, and R_B correspond to the yellow, green, or red button being pressed, respectively. Because both agents A_2 and A_3 must simultaneously stand on the red button for the event R_B to occur, we additionally include the events $A_2^{R_B}, A_2^{\neg R_B}, A_3^{R_B}, A_3^{\neg R_B}$ in Σ, which represent A_2 or A_3 individually either pressing or not pressing the red button. The event *Goal* corresponds to agent A_1 successfully reaching the goal location.

The states $u \in U$ of the RM represent different stages of the team's task. Transitions between RM states are triggered by events from Σ. For example, the buttons task starts in state u_I. From this state, no buttons have been pressed and so the colored regions cannot be entered. When A_1 presses the yellow button in the environment, the event Y_B will cause the RM to transition to state u_1. From this state, only event G_B causes an outgoing transition; A_2 may now proceed across the yellow region in order to press the green button. Because, for example, the red region still prevents A_1 from reaching the goal location, event *Goal* does not cause a transition from u_1. In this way, transitions in the RM represent progress through the task.

Output function σ assigns reward values to these transitions. In this work, we restrict ourselves to *task completion* RMs, similar to those studied in [41]; σ should only reward transitions that result in the immediate completion of the task. To formalize this idea, we define a subset $F \subseteq U$ of reward states. If the RM is in a state belonging to F, it means the task is complete. The output function is then defined such that $\sigma(u, u') = 1$ if $u \notin F$ and $u' \in F$, and is defined to output 0 otherwise. Furthermore, there should be no outgoing transitions from states in F. In Figure 1b, $F = \{u_7\}$. If the event $Goal \in \Sigma$ occurs while the RM is in state u_6 the task has been successfully completed; the RM will transition and return reward 1.

A run of RM $\mathcal{R}$ on the sequence of events $e_0e_1...e_k \in \Sigma^*$ is a sequence $u_0e_0u_1e_1...u_ke_ku_{k+1}$, where $u_0 = u_I$ and $u_{t+1} = \delta(u_t, e_t)$. If $u_{k+1} \in F$, then $\sigma(u_k, u_{k+1}) = 1$. In this case we say that the event sequence $e_0...e_k$ completes the task described by $\mathcal{R}$, and we denote this statement $\mathcal{R}(e_0...e_k) = 1$. Otherwise, $\mathcal{R}(e_0...e_k) = 0$. For example, $\mathcal{R}(Y_BG_BA_2^{R_B}A_3^{R_B}R_BGoal) = 1$, but $\mathcal{R}(Y_BG_BA_2^{R_B}) = 0$.

For notational convenience, we define the transition from state u under an event sequence $e_0e_1...e_k \in \Sigma^*$ using the recursive definitions $\delta(u, \varepsilon) = u$ and $\delta(u, \xi e) = \delta(\delta(u, \xi), e)$ for $\xi \in \Sigma^*$ and $e \in \Sigma$. Here, Σ^* is the Kleene closure of Σ and ε is the empty string of events. So, $\mathcal{R}(e_0e_1...e_k) = 1$ if and only if $\delta(u_I, e_0e_1...e_k) \in F$.



(a) Cooperative buttons domain. **(b) Reward machine encoding the cooperative buttons task.**

Figure 1: The multi-agent buttons task. In Figure (a), the colored circles denote the locations of the buttons, the thick black areas are walls the agents cannot cross, and the numbered dotted lines show the order of high-level steps necessary to complete the task. The set of events of the RM in (b) is $\Sigma = \{Y_B, G_B, R_B, A_2^{R_B}, A_2^{\neg R_B}, A_3^{R_B}, A_3^{\neg R_B}, Goal\}$.

3.2 Labeling Functions and Q-Learning with Reward Machines

RMs may be applied to RL problems by using them to replace the reward function in an MDP. However, RMs describe tasks in terms of abstract events. To allow an RM to interface with the underlying environment, we define a *labeling function* $L : S \times U \rightarrow 2^{\Sigma}$, which abstracts the current environment state to sets of high-level events. Note that L takes the current RM state $u \in U$ as well as the environment state $s \in S$ as input, allowing the events output by L to depend not only on the environment, but also on the current progress through the task. This component of L's definition aides in specifying local labeling functions, discussed in §4.1. Also, L is defined to output collections of events, allowing it to capture scenarios in which multiple events occur concurrently. In such a scenario, the events are passed as a sequence to the RM in no particular order.

Q-learning with RMs (QRM) [14] is an algorithm that learns a collection of q-functions, one for each RM state $u \in U$, corresponding to the optimal policies for each stage of the task. Algorithm 1 details the method. At each time step, if the agent is in RM state u_1 and environment state s_1, it uses its estimate of $q_{u_1}(s_1, \cdot)$ to select action a. The environment accordingly progresses to state s_2. The events output by $L(s_2, u_1)$ then cause the RM to transition to state u_2, and the corresponding reward output by σ is used to update the estimate of $q_{u_1}(\mathbf{s}_1, a)$: the optimal q-function for RM state u_1. At this stage, the agent also queries the rewards and RM transitions that would have occurred had the RM been in any other state u. This counterfactual information is used to update the estimate of each q-function q_u. The tabular QRM algorithm is guaranteed to converge to an optimal policy [14].

A naive approach to applying RMs in the MARL setting would be to treat the entire team as a single agent and to use QRM to learn a centralized policy. This approach quickly becomes intractable, however, due to the exponential scaling of the number of states and actions with the number of agents. Furthermore, it assumes agents communicate with a central controller at every time step, which may be undesirable from an implementation standpoint.

Algorithm 1: Q-Learning with Reward Machines

Input: $\mathcal{R} = \langle U, u_I, \Sigma, \delta, \sigma, F \rangle$, L, γ, α
Output: $Q = \{q_u : S \times A \rightarrow \mathbb{R} | u \in U\}$

$Q \leftarrow InitializeQFunctions()$
for $n = 1$ ***to*** $NumEpisodes$ **do**
  $u_1 \leftarrow u_I$, $s_1 \leftarrow environmentInitialState()$
  for $t = 0$ ***to*** $NumSteps$ **do**
    $a \leftarrow getAction(q_{u_1}, s_1)$
    $s_2 \leftarrow executeAction(s_1, a)$
    $r, u_2 \leftarrow rewardMachineOutput(u_1, L(s_2, u_1))$
    $q_{u_1}(s_1, a) \leftarrow$
     $(1 - \alpha) q_{u_1}(s_1, a) + \alpha(r + \gamma \max_{a' \in A} q_{u_2}(s_2, a'))$
    for $u \in U, u \neq u_1$ **do**
      $r, u' \leftarrow rewardMachineOutput(u, L(s_2, u))$;
      $q_u(s_1, a) \leftarrow$
       $(1 - \alpha) q_u(s_1, a) + \alpha(r + \gamma \max_{a' \in A} q_{u'}(s_2, a'))$
    $u_1 \leftarrow u_2$, $s_1 \leftarrow s_2$
    if $u_1 \in F$ **then**
      break
return Q

3.3 Team Task Decomposition

A decentralized approach to MARL treats the agents as individual decision-makers, and therefore requires further consideration of the information available to each agent. In this work, we assume the i^{th} agent can observe its own local state $s_i \in S_i$, but not the local states of its teammates. Given an RM $\mathcal{R}$ describing the team's task and the corresponding event set Σ, we assign the i^{th} agent a subset $\Sigma_i \subseteq \Sigma$ of events. These events represent the high-level information that is available to the agent. We call Σ_i the *local event set* of agent i. We assume that all events represented in Σ belong to the local event set of at least one of the agents, thus $\bigcup_{i=1}^{N} \Sigma_i = \Sigma$.

For example, in the three-agent buttons task, the local event set assigned to A_1 is $\Sigma_1 = \{Y_B, R_B, Goal\}$; A_1 has access to the yellow button, must know when the red button has been pressed, and

should eventually proceed to the goal location. Note, for example, that events $A_2^{R_B}$ and $A_2^{\neg R_B}$ are not included in Σ_1 because they are specific to agent A_2; they are not necessary pieces of information for the completion of A_1's sub-task. Similarly, event G_B is not in Σ_1 because from the perspective of A_1's task, it is also only an intermediate step in the process of pressing the red button. The event sets of A_2 and A_3 are $\Sigma_2 = \{Y_B, G_B, A_2^{R_B}, A_2^{\neg R_B}, R_B\}$ and $\Sigma_3 = \{G_B, A_3^{R_B}, A_3^{\neg R_B}, R_B\}$, respectively.

Extending the definition of natural projections on automata [16, 40] to reward machines, for each agent i, we define a new RM, $\mathcal{R}_i = \langle U_i, u_I^i, \Sigma_i, \delta_i, \sigma_i, F_i \rangle$, called the projection of $\mathcal{R}$ onto Σ_i. We begin by defining a notion of state equivalence under event set Σ_i

Given $\mathcal{R} = (U, u_I, \Sigma, \delta, \sigma, F)$ and $\Sigma_i \subseteq \Sigma$ we define the equivalence relation on states under event set Σ_i, denoted $\sim_i \subseteq U \times U$, as the smallest equivalence relation such that:

(1) For all $u_1, u_2 \in U$, and $e \in \Sigma$,
if $u_2 = \delta(u_1, e)$ and $e \notin \Sigma_i$, then $(u_1, u_2) \in \sim_i$.
(2) If $u_1, u_1' \in U$, $(u_1, u_1') \in \sim_i$ and $\delta(u_1, e) = u_2$, $\delta(u_1', e) = u_2'$ are both defined for some $e \in \Sigma_i$, then $(u_2, u_2') \in \sim_i$.

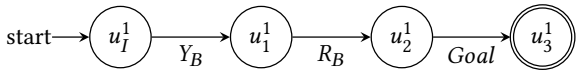
The equivalence class of any state $u \in U$ under equivalence relation $\sim_i$ is $[u]_i = \{u' \in U | (u, u') \in \sim_i\}$. The quotient set of U by $\sim_i$ is defined as the set of all equivalence classes $U/\sim_i = \{[u]_i | u \in U\}$. Equivalence relation $\sim_i$ may be computed by first finding the smallest equivalence relation satisfying condition (1), and then applying at most $|U|$ state-merging steps in order to satisfy (2). An algorithm to compute this equivalence relation with runtime $O(|U|^7|\Sigma_i|^2 + |U|^5|\Sigma_i| + |U|^4)$ is provided in appendix C of [40].

In words, the first condition guarantees that two states $u_1, u_2 \in U$ of the RM $\mathcal{R}$ are members of the same equivalence class if a transition exists between them that is triggered by an event outside of the local event set Σ_i. The equivalence classes thus represent the collections of states of $\mathcal{R}$ that are indistinguishable to an agent who may only observe events from Σ_i. The second condition ensures that from any equivalence class, a particular event $e \in \Sigma_i$ may only trigger transitions to a unique successor equivalence class. Using this equivalence relation, we define the projection of $\mathcal{R}$ onto Σ_i.

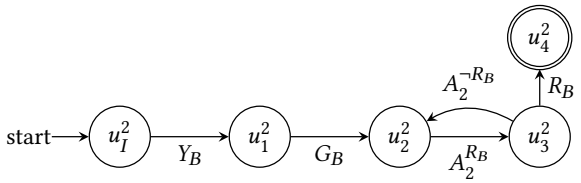
Definition 2. *(RM projection onto a local event set) Given a reward machine $\mathcal{R} = (U, u_I, \Sigma, \delta, \sigma, F)$ and a local event set $\Sigma_i \subseteq \Sigma$, we define the projection of $\mathcal{R}$ onto Σ_i as $\mathcal{R}_i = (U_i, u_I^i, \Sigma_i, \delta_i, \sigma_i, F_i)$.*

- *The set of projected states U_i is given by $U/\sim_i$; each state $u^i \in U_i$ is an equivalence class of states from $u \in U$.*
- *The initial state is $u_I^i = [u_I]_i$.*
- *Transition function $\delta_i : U_i \times \Sigma_i \to U_i$ is defined such that $u_2^i = \delta_i(u_1^i, e)$ if and only if there exist $u_1, u_2 \in U$ such that $u_1^i = [u_1]_i$, $u_2^i = [u_2]_i$, and $u_2 = \delta(u_1, e)$.*
- *The projected set of final states is defined as $F_i = \{u^i \in U_i | \exists u \in F$ such that $u^i = [u]_i\}$.*
- *The output function $\sigma_i : U_i \times U_i \to \mathbb{R}$ is defined such that $\sigma_i(u_1^i, u_2^i) = 1$ if $u_1^i \notin F_i, u_2^i \in F_i$ and $\sigma(u_1^i, u_2^i) = 0$ otherwise.*

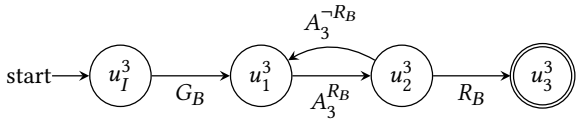
The intuition behind this definition is as follows. To define the new set of states, we remove all transitions triggered by events *not* contained in Σ_i and merge the corresponding states. The remaining transitions are used to define the projected transition function δ_i. The reward states F_i are defined as the collection of merged states containing at least one reward state $u \in F$ from the original RM, and output function σ_i is defined accordingly. We note that δ_i is a well-defined function as a result of condition (2) in the definition of the equivalence relation $\sim_i$.



(a) RM projection onto local event set $\Sigma_1 = \{Y_B, R_B, Goal\}$.



(b) RM projection onto local event set $\Sigma_2 = \{Y_B, G_B, A_2^{R_B}, A_2^{\neg R_B}, R_B\}$.



(c) RM projection onto local event set $\Sigma_3 = \{G_B, A_3^{R_B}, A_3^{\neg R_B}, R_B\}$.

Figure 2: Projections of the team RM illustrated in Figure 1.

Figures 2a, 2b, and 2c show the results of projecting the task RM from Figure 1b onto the local event sets of Σ_1, Σ_2, and Σ_3 respectively. As an example, we specifically examine $\mathcal{R}_1 = \langle U_1, u_I^1, \Sigma_1, \delta_1, \sigma_1, F_1 \rangle$, illustrated in in Figure 2a. Because events G_B, $A_2^{R_B}$, $A_2^{\neg R_B}$, $A_3^{R_B}$, and $A_3^{\neg R_B}$ are not elements of Σ_1, any states connected by these events are merged to form the projected states U_1. For example, states $u_1, u_2, u_3, u_4, u_5 \in U$ which comprise the diamond structure in Figure 1b, are all merged into projected state $u_1^1 \in U_1$. Intuitively, this portion of the team's RM $\mathcal{R}$ encodes the necessary coordination between A_2 and A_3 to press the red button, which is irrelevant to A_1's portion of the task and is thus represented as a single state in $\mathcal{R}_1$. Note that $\mathcal{R}_1$ describes A_1's contribution to the team's task; press the yellow button then wait for the red button to be pressed, before proceeding to the goal location. Intuitively, this high-level behavior is correct with respect to the team task, regardless of the behavior A_2 or A_3.

However, clearly not all RMs and local event sets lead to projections describing behavior compatible with the original task. For example, if $Y_B \notin \Sigma_1$, then $\mathcal{R}_1$ would instead describe a task in which agent A_1 is not required to press the yellow button. This conflicts with the original task, in which the yellow button must be pressed before agent A_2 can proceed across the yellow region. An important notion to define then, is compatibility between the original task, and the task described by a collection of projected RMs.

Consider some finite event sequence $e_0 ... e_k \in \Sigma^*$. The natural projection [22] of the sequence onto onto Σ_i^*, denoted $P_i(e_0 ... e_k) \in \Sigma_i^*$, is defined recursively by the relationships $P_i(\varepsilon) = \varepsilon$, $P_i(\xi e) = P_i(\xi)e$ if $e \in \Sigma_i$, and $P_i(\xi e) = P_i(\xi)$ if $e \notin \Sigma_i$ for any $\xi \in \Sigma^*$. We remark that $P_i(\xi)$ may be thought of as the event sequence ξ from the point of view of the i^{th} agent.

Theorem 1 defines a condition guaranteeing that the composition of the individual behaviors described by the projected RMs is equivalent to the behavior described by the original team RM. This condition uses the notions of parallel composition and bisimilarity.

Intuitively, the parallel composition of two or more RMs is a new RM describing all the possible interleavings of their events. If each RM encodes a sub-task, their parallel composition encodes all possible results of carrying out those sub-tasks concurrently. The bisimilarity of two RMs ensures that every sequence of transitions and rewards from one RM can be matched by the other, and vice versa; it provides a formal notion of equivalence between the tasks they encode. These are common concepts for finite transition systems [1, 5, 20], that are formally defined for RMs in the supplementary material. The supplementary material may be found in the extended version of this paper [28].

Theorem 1. *Given RM $\mathcal{R}$ and a collection of local event sets $\Sigma_1, \Sigma_2, ..., \Sigma_N$ such that $\bigcup_{i=1}^{N} \Sigma_i = \Sigma$, let $\mathcal{R}_1, \mathcal{R}_2, ..., \mathcal{R}_N$ be the corresponding collection of projected RMs. Suppose $\mathcal{R}$ is bisimilar to the parallel composition of $\mathcal{R}_1, \mathcal{R}_2, ..., \mathcal{R}_N$. Then given an event sequence $\xi \in \Sigma^*$, $\mathcal{R}(\xi) = 1$ if and only if $\mathcal{R}_i(P_i(\xi)) = 1$ for all $i = 1, 2, ..., N$. Otherwise, $\mathcal{R}(\xi) = 0$ and $\mathcal{R}_i(P_i(\xi)) = 0$ for all $i = 1, 2, ..., N$.*

Proof. Let $\mathcal{R}_p = \langle U_p, u_I^p, \Sigma, \delta_p, \sigma_p, F_p \rangle$ be the parallel composition of $\mathcal{R}_1, ..., \mathcal{R}_N$. Note that $\mathcal{R}(\xi) = 1$ if and only if $\delta(u_I, \xi) \in F$. Similarly, $\mathcal{R}_i(P_i(\xi)) = 1$ if and only if $\delta_i(u_I^i, P_i(\xi)) \in F_i$ for every $i = 1, ..., N$. So, it is sufficient to show that $\delta(u_I, \xi) \in F$ if and only if $\delta_i(u_I^i, P_i(\xi)) \in F_i$ for every $i = 1, ..., N$. Given the assumption that $\mathcal{R}$ is bisimilar to $\mathcal{R}_p$, we can show by induction that $\delta(u_I, \xi) \in F$ if and only if $\delta_p(u_I^p, \xi) \in F_p$ (see chapter 7 of [1]). Now, given the definition of parallel composition, it is readily seen that $\delta_p(u_I^p, \xi) \in F_p$ if and only if $\delta_i(u_I^i, P_i(\xi)) \in F_i$ for every $i = 1, ..., N$ [20]. □

We note that [16, 17] present conditions, in terms of $\mathcal{R}$ and Σ_1, ..., Σ_N, which may be applied to check whether $\mathcal{R}$ is bisimilar to the parallel composition of its projections $\mathcal{R}_1, ..., \mathcal{R}_N$. Alternatively, one may computationally check whether this result holds by automatically constructing the parallel composition of the projected RMs and applying the *Hopcroft-Karp* algorithm to check bisimilarity [2, 13]. The runtime of this algorithm is $O(|\Sigma|(|U| + |U_p|))$, where U are the states of $\mathcal{R}$, and U_p are the states of the parallel composition of $\mathcal{R}_1, ..., \mathcal{R}_N$ [13]. If the bisimilarity condition does not hold, the task designer might add to the local event sets, giving the agents access to more information, and re-check the condition.

4 DECENTRALIZED Q-LEARNING WITH PROJECTED REWARD MACHINES (DQPRM)

Inspired by Theorem 1, we propose a distributed approach to learning a decentralized policy. Our idea is to use the projected RMs to define a collection of single-agent RL tasks, and to train each agent on their respective task using the QRM algorithm described in §3.2.

For clarity, we wish to train the agents using their projected RMs in an *individual setting*: the agents take actions in the environment in the absence of their teammates. However, the policies they learn should result in a team policy that is successful in the *team setting*, in which the agents interact simultaneously with the shared environment.

4.1 Local Labeling Functions and Shared Event Synchronization

Projected reward machine $\mathcal{R}_i$ defines the task of the i^{th} agent in terms of high-level events from Σ_i. As discussed in §3.2, to connect $\mathcal{R}_i$ with the underlying environment, a labeling function is required to define the environment states that cause the events in Σ_i to occur. In the team setting, we can intuitively define a labeling function $L : \mathcal{S} \times U \to 2^{\Sigma}$ mapping team states $\mathbf{s} \in \mathcal{S}$ and RM states $u \in U$ to sets of events. For example, $L(\mathbf{s}, u) = \{R_B\}$ if $\mathbf{s}$ is such that A_2 and A_3 are pressing the red button and $u = u_5$.

To use $\mathcal{R}_i$ in the individual setting however, we must first define a *local labeling function* $L_i : S_i \times U_i \to 2^{\Sigma_i}$ mapping the local states $s_i \in S_i$ and projected RM states $u^i \in U_i$ to sets of events in Σ_i. Operating under the assumption that only one event can occur per agent per time step, we require that, for any local state pair (s_i, u^i), $L_i(s_i, u^i)$ returns at most a single event from Σ_i. Furthermore, the local labeling functions L_1, L_2,..., L_N should be defined such that they always collectively output the same set of events as L, when being used in the team setting.

For a given event $e \in \Sigma$, we define the set $I_e = \{i | e \in \Sigma_i\}$ as the *collaborating agents on e*.

Definition 3. *(Decomposable labeling function) A labeling function $L : \mathcal{S} \times U \to 2^{\Sigma}$ is considered decomposable with respect to local event sets $\Sigma_1, \Sigma_2, ..., \Sigma_N$ if there exists a collection of local labeling functions $L_1, L_2, ..., L_N$ with $L_i : S_i \times U_i \to 2^{\Sigma_i}$ such that:*

1. *$|L_i(s_i, u^i)| \leq 1$ for every $s_i \in S_i$ and every $u^i \in U_i$.*
2. *$L(s, u)$ outputs event e if and only if $L_i(s_i, u^i)$ outputs event e for every i in I_e. Here, s_i is the i^{th} component of the team's joint state $\mathbf{s}$, and $u^i \in U_i$ is the state of RM $\mathcal{R}_i$ containing state $u \in U$ from RM $\mathcal{R}$ (recall that states in U_i correspond to collections of states from U).*

Note that L will be decomposable if we can define L_1, ..., L_N to satisfy the conditions in Definition 3. Following this idea, we conceptually construct L_i from L as follows: $L_i(\bar{s}_i, u^i)$ outputs event $e \in \Sigma_i$ whenever there exists a possible configuration of agent i's teammates $\mathbf{s} = (s_1, ..., \bar{s}_i, ..., s_N)$ such that $L(\mathbf{s}, u)$ outputs e, where $u \in U$ is any state belonging to $u^i \in U_i$. Our interpretation of this definition of L_i is as follows. While $L(\mathbf{s}, u)$ outputs the events that occur when the team is in joint state $\mathbf{s}$ and RM state u, the local labeling function $L_i : S_i \times A_i \to 2^{\Sigma_i}$ outputs the events in Σ_i that *could* be occurring from the point of view of an agent who knows L, but may only observe $s_i \in S_i$ and $u^i \in U_i$.

Furthermore, to ensure L_1, ..., L_N output an event e only if L does, we also require that each event $e \in \Sigma$ be "under the control of at least one of the agents" in the following sense: if the agent is not in some particular subset of local states, the event will not be returned by L, regardless of the states of the agent's teammates. A more formal definition of this construction of L_i as well as conditions on L that ensure L_i are well defined, are given in the supplementary material [28].

We say event $e \in \Sigma$ is a *shared event* if it belongs to the local event sets of multiple agents, i.e., if $|I_e| > 1$. In the buttons task, $Y_B \in \Sigma_1 \cap \Sigma_2$ is an example of a shared event. Suppose A_1 and A_2 use the events output by L_1 and L_2, respectively, to update $\mathcal{R}_1$ and $\mathcal{R}_2$, while interacting in the team setting. Because Σ_1 and Σ_2 both include

the event Y_B, the agents must *syncrhonize* on this event: Y_B should simultaneously cause transitions in both projected RMs, or it should cause a transition in neither of them. In practice, synchronization on shared events is implemented as follows: If L_i returns a shared event e, the i^{th} agent should check with all teammates in I_e whether their local labeling functions also returned e, before using the event to update $\mathcal{R}_i$. Event synchronization corresponds to the agents communicating and collectively acknowledging that the shared event occurred, before progressing through their repsective tasks.

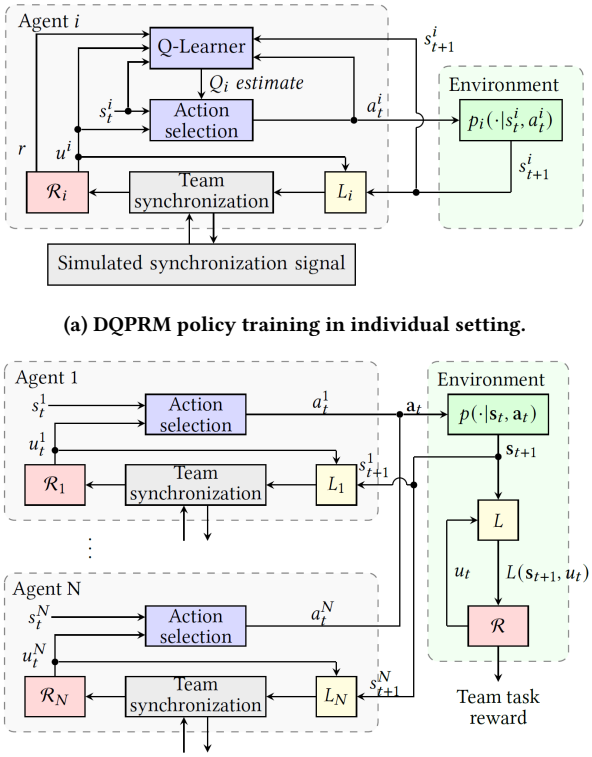
Given a sequence of joint states $\mathbf{s}_0\mathbf{s}_1...\mathbf{s}_k$ in the team setting, we may use L and $\mathcal{R}$ to uniquely define a corresponding sequence $L(\mathbf{s}_o...\mathbf{s}_k) \in \Sigma^*$ of events. Similarly, given the corresponding collection of sequences $\{s_0^i...s_k^i\}_{i=1}^N$ of local states, local labeling functions L_1, ..., L_N, and assuming that the agents synchronize on shared events, we may define the corresponding sequences $L_i(s_0^i...s_k^i) \in \Sigma_i^*$ of events for every $i = 1, 2, ..., N$. A step-by-step construction of sequences $L(\mathbf{s}_0...\mathbf{s}_k)$ and $L_i(s_0^i...s_k^i)$, as well as the details of the induction step of the proof of Theorem 2 are provided in the supplementary material [28].

Theorem 2. *Given $\mathcal{R}$, L, and Σ_1, ..., Σ_N, suppose the bisimilarity condition from Theorem 1 holds. Furthermore, assume L is decomposable with respect to Σ_1, ..., Σ_N with the corresponding local labeling functions L_1, ..., L_N. Let $\mathbf{s}_0...\mathbf{s}_k$ be a sequence of joint environment states and $\{s_0^i...s_k^i\}_{i=1}^N$ be the corresponding sequences of local states. If the agents synchronize on shared events, then $\mathcal{R}(L(\mathbf{s}_0...\mathbf{s}_k)) = 1$ if and only if $\mathcal{R}_i(L_i(s_0^i...s_k^i)) = 1$ for all $i = 1, 2, ..., N$. Otherwise $\mathcal{R}(L(\mathbf{s}_0...\mathbf{s}_k)) = 0$ and $\mathcal{R}_i(L_i(s_0^i...s_k^i)) = 0$ for all $i = 1, 2, ..., N$.*

Proof. Note that given the result of Theorem 1, it is sufficient to show that for every $i = 1, 2, ..., N$, the relationship $P_i(L(\mathbf{s}_0\mathbf{s}_1...\mathbf{s}_k)) = L_i(s_0^i s_1^i...s_k^i)$ holds. In words, we wish to show that for every i, the projection of the sequence output by labeling function L is equivalent to the sequence of synchronized outputs of local labeling function L_i. To do this, we show that $l_t \cap \Sigma_i = \tilde{l}_t^i$ for every $i = 1, ..., N$ and every $t = 1, ..., k$. Here, l_t denotes the output of labeling function L at time t, and $\tilde{l}_t^i$ denotes the synchronized output of local labeling function L_i at time t.

At time $t = 0$, $l_0, \tilde{l}_0^i$ are defined to be the empty set and the above condition holds trivially. We also have that at time $t = 0$, $u_I^i \in u_I$ by definition of initial state u_I^i. If, for any time t, $u_t \in u_t^i$ for each $i = 1, 2, ..., N$, we can use the definition of the decomposability of labeling function L to show that $l_{t+1} \cap \Sigma_i = \tilde{l}_{t+1}^i$ for each i. Using the definition of the projected RMs, we may show that $l_{t+1} \cap \Sigma_i = \tilde{l}_{t+1}^i$ implies $u_{t+1} \in u_{t+1}^i$. Thus by induction, we conclude the proof. □

Let $V^{\boldsymbol{\pi}}(\mathbf{s})$ denote the expected sum of future undiscounted rewards returned by $\mathcal{R}$, given the team follows joint policy $\boldsymbol{\pi}$ from joint environment state $\mathbf{s} \in \mathcal{S}$ and initial RM state u_I. Similarly, let $V_i^{\boldsymbol{\pi}}(\mathbf{s})$ denote the expected future reward returned by $\mathcal{R}_i$, given the team follows the same policy from state $\mathbf{s}$ and projected initial RM state u_I^i. Using the result of Theorem 2 and the Fréchet conjunction inequality, we provide the following upper and lower bounds on the value function corresponding to $\mathcal{R}$, in terms of the value functions corresponding to projected RMs $\mathcal{R}_i$.



(a) DQPRM policy training in individual setting.

(b) DQPRM policy execution in team setting.

Figure 3

Theorem 3. *If the conditions in Theorem 2 are satisfied, then*

$$\max\{0, V_1^{\boldsymbol{\pi}}(\mathbf{s}) + V_2^{\boldsymbol{\pi}}(\mathbf{s}) + ... + V_N^{\boldsymbol{\pi}}(\mathbf{s}) - (N-1)\} \leq V^{\boldsymbol{\pi}}(\mathbf{s}) \leq \min\{V_1^{\boldsymbol{\pi}}(\mathbf{s}), V_2^{\boldsymbol{\pi}}(\mathbf{s}), ..., V_N^{\boldsymbol{\pi}}(\mathbf{s})\}.$$

Proof. Note that because $\mathcal{R}$ returns 1 if the task it encodes is completed and 0 otherwise, $V^{\boldsymbol{\pi}}(\mathbf{s}_0)$ is equivalent to the probability of the task being completed, given the team follows policy $\boldsymbol{\pi}$ from the initial state $\mathbf{s}_0$ and initial reward machine state u_I. Similarly, $V_i^{\boldsymbol{\pi}}(\mathbf{s}_0)$ is the probability of satisfying the task encoded by $\mathcal{R}_i$.

By the result of Theorem 2, for any sequence of team states $\mathbf{s}_0...\mathbf{s}_k$, $\mathcal{R}(L(\mathbf{s}_0...\mathbf{s}_k)) = 1$ if and only if $\mathcal{R}_i(L_i(s_1^i...s_k^i)) = 1$ for all $i = 1, ..., N$. So, the likelihood of completing the task encoded by $\mathcal{R}$ under policy $\boldsymbol{\pi}$ is equivalent to the likelihood of simultaneously completing all the tasks encoded by the collection $\{\mathcal{R}_i\}_{i=1}^N$ under the same policy. Using our interpretation of $V^{\boldsymbol{\pi}}(\mathbf{s}_0)$ and $V_i^{\boldsymbol{\pi}}(\mathbf{s}_0)$ as these probabilities, we apply the Fréchet conjunction inequality to arrive at the final result. □

4.2 Training and Evaluating

Theorem 2 tells us that it makes no difference whether we use RM $\mathcal{R}$ and team labeling function L, or projected RMs $\mathcal{R}_1$,...,$\mathcal{R}_N$ and local labeling functions L_1, ..., L_N to describe the team task. By replacing $\mathcal{R}$ and L with $\mathcal{R}_1$, ..., $\mathcal{R}_N$ and L_1, ..., L_N however, we note that the only interactions each agent has with its teammates are synchronizations on shared events.

This key insight provides a method to train the agents separately from their teammates. We train each agent in an individual setting, isolated from its teammates, using rewards returned from $\mathcal{R}_i$ and events returned from L_i. Whenever L_i outputs what would be a shared event in the team setting, we randomly provide a simulated synchronization signal with a fixed probability of occurrence. Figure 3a illustrates this approach.

By simulating the synchronization on shared events, we take an optimistic approach to decentralized learning. Each agent learns to interact with idealized teammates in the sense that during training, any shared events necessary for task progression will always occur, albeit after some random amount of time.

During training, each agent individually performs q-learning to find an optimal policy for the sub-task described by its projected RM, similarly to as described in §3.2. The i^{th} agent learns a collection of q-functions $Q_i = \{q_{u^i} | u^i \in U_i\}$ such that each q-function $q_{u^i} : S_i \times A_i \to \mathbb{R}$ corresponds to the agent's optimal policy while it is in projected RM state u^i.

To evaluate the learned policies, we test the team by allowing the agents to interact in the team environment and evaluate the team's performance using team task RM $\mathcal{R}$. Each agent tracks its own task progress using its projected RM $\mathcal{R}_i$ and follows the policy it learned during training, as shown in Figure 3b.

5 EXPERIMENTAL RESULTS

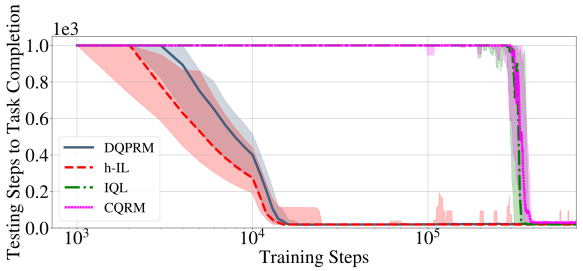
In this section, we provide empirical evaluations of DQPRM in three task domains.[1] The buttons task is as described in §3. We additionally consider two-agent and ten-agent rendezvous tasks in which each agent must simultaneously occupy a specific rendezvous location before individually navigating to separate goal locations.

We compare DQPRM's performance against three baseline algorithms: the naive centralized QRM (CQRM) algorithm described in §3.2, independent q-learners (IQL) [34], and hierarchical independent learners (h-IL) [35]. Because of the non-Markovian nature of the tasks, we provide both the IQL and h-IL agents with additional memory states. In the buttons task, the memory states encode which buttons have already been pressed. In the rendezvous task, the memory state encodes whether or not the team has completed the rendezvous, and whether each agent has reached its goal.

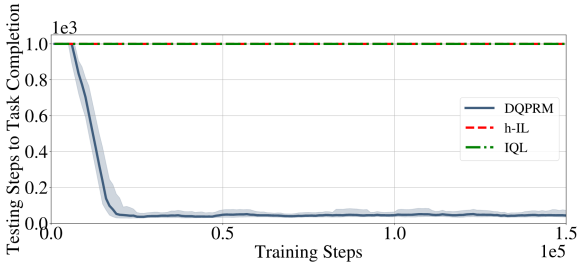
Each IQL agent learns a q-function mapping augmented state-action pairs to values. That is, the i^{th} agent learns a q-function $q_i : S_i \times S_{M_i} \times A_i \to \mathbb{R}$, where S_i, A_i are the local states and actions of the agent and S_{M_i} is the finite set of its memory states.

Our implementation of h-IL is inspired by the learning structure outlined in [35]. Each agent uses tabular q-learning to learn a meta-policy — which uses the current memory state to select a high-level option — as well as a collection of low-level policies — which implement those options in the environment. The available options correspond to the high-level tasks available to each agent. For example, A_1 in the buttons task is provided with the following three options: remain in a non-colored region, navigate to the yellow button, and navigate to the goal location. Furthermore, we prune the available options when necessary. For example, before the red button has been pressed, A_1 cannot cross the red region to reach the goal, so, it doesn't have access to the corresponding option.

[1]Project code is publicly available at: github.com/cyrusneary/rm-cooperative-marl.



(a) Two-agent rendezvous task.



(b) Ten-agent rendezvous task.

(c) Three-agent buttons task.

Figure 4: Algorithm performance on various tasks. Lower is better. The y-axis show the number of steps required for the learned policies to complete the task. The x-axis shows the number of elapsed training steps.

In all algorithms, we use a discount factor $\gamma = 0.9$ and a learning rate $\alpha = 0.8$. For action selection, we use softmax exploration with a constant temperature parameter $\tau = 0.02$ [36]. In the training of the DQPRM agents, if an agent observes a shared event, then with probability 0.3, it is provided with a signal simulating successful synchronization with all collaborators.

All tasks are implemented in a 10x10 gridworld and all agents have the following 5 available actions: move right, move left, move up, move down, or don't move. If an agent moves in any direction, then with a 2% chance the agent will instead slip to an adjacent state. Each episode lasts 1,000 time steps. We perform periodic testing episodes in which the agents exploit the policies they have learned and team's performance is recorded.

Figure 4 shows the experimental results for each algorithm over 10 separate runs per domain. The figures plot the median number of testing steps required to complete the team task against the number of elapsed training steps. Because DQPRM trains each agent individually, one training step on the plot refers to one training

step taken by each individual agent. The shaded regions enclose the 25^{th} and 75^{th} percentiles. We note that the CQRM baseline is only tested in the two-agent scenario because the centralized approach requires excessive amounts of memory for more agents; storing a centralized policy for three agents requires approximately two billion separate values.

While the h-IL baseline marginally outperforms the proposed DQPRM algorithm in the two-agent rendezvous task, in the more complex tasks involving more agents, DQPRM outperforms all baseline methods. In the ten-agent rendezvous task, the baseline methods fail to learn a successful team policy within the allowed number of training steps. This demonstrates the ability of the proposed DQPRM algorithm to scale well with the number of agents. In the buttons task, DQPRM quickly learns a policy that completes the task 20 steps faster than that of the h-IL baseline. This difference in performance likely arises because the hierarchical approach has pruned optimal policies, whereas QRM guarantees that each agent converges to the optimal policy for their sub-task [14].

The key advantage of the DQPRM algorithm, is that it uses the information provided in the RM to train the agents entirely separately. This removes the problem of non-stationarity and it allows each agent to more frequently receive reward during training. This is especially beneficial to the types of tasks we study, which have sparse and delayed feedback. We further discuss the differences between DQPRM and hierarchical approaches to MARL in §6.

6 RELATED WORK

Task decomposition in multi-agent systems has been studied from a planning and cooperative control perspective [7, 16–18]. These works examine the conditions in which group tasks described by automata may be broken into sub-tasks executable by individuals. [6, 8] provide methods to synthesize control policies for large-scale multi-agent systems with temporal logic specifications. However, all of these works assume a known model of the environment, differing from the learning setting of this paper.

Several works have explored combining formal methods and MARL. Recently, the authors of [21] present Extended Markov Games; a mathematical model allowing multiple agents to concurrently learn to satisfy multiple non-Markovian task specifications. The authors of [27] use minimax deep q-learning to solve zero-sum adversarial games in which the reward is encoded by signal temporal logic specifications. However, to the best of our knowledge, none have yet studied how automata-based task descriptions can be used to decompose a cooperative problem in a way that allows the agents to learn in the absence of their teammates.

The MARL literature is rich [11, 12, 26, 42]. A popular popular approach is IQL [34]; each agent learns independently and treats its teammates as part of the environment. [10, 19, 37] decompose cooperative tasks by factoring the joint q-function into components.

More recently, centralized training decentralized execution (CTDE) paradigm algorithms, such as QMIX [29], have shown empirical success in cooperative deep MARL problems [24, 29, 31, 32]. These algorithms enforce the assumption that the team's q-function can be decomposed in a way that allows the agents to make decentralized decisions. CTDE algorithms have been shown to perform well on challenging tasks [30]. However, the centralized training required by these methods can be sample inefficient, as observed in the results of our centralized approach to learning with reward machines, shown in Figure 4a. Our work studies how take advantage of tasks in which only sparse interactions are required between the agents, to avoid simultaneous training of the agents altogether.

Our work, which examines cooperative tasks that have sparse and temporally delayed rewards, is most closely related to hierarchical approaches to MARL. In particular, [9, 25] use task hierarchies to decompose the multi-agent problem. By learning cooperative strategies only in terms of the sub-tasks at the highest levels of the hierarchy, agents learn to coordinate much more efficiently than if they were sharing information at the level of primitive state-action pairs. More recently, [35] empirically demonstrates the effectiveness of a *deep* hierarchical approach to certain cooperative MARL tasks. A key difference between task hierarchies and RMs, is that RMs explicitly encode the temporal ordering of the high-level sub-tasks. It is by taking advantage of this information that we are able to break a team's task into components, and to train the agents independently while guaranteeing that they are learning behavior appropriate for the original problem. Conversely, in a hierarchical approach, the agents must still learn to coordinate at the level of sub-tasks. Thus, the learning problem remains inherently multi-agent, albeit simplified.

In this work, we assume the task RM is known, and present a method to use its decomposition to efficiently solve the MARL problem. The authors of [15, 41] demonstrate that, in the single-agent setting, RMs can be learned from experience, removing the assumption of the RM being known *a priori* by the learner. This presents an interesting direction for future research: how may agents learn, in a multi-agent setting, RMs encoding either the team's task or projected local tasks. Furthermore, [4, 14] demonstrate in the single-agent setting that RMs may be applied to continuous environments by replacing tabular q-learning with double deep q-networks [38]. This extension to more complex environments also readily applies to our work, which decomposes multi-agent problems into collections of RMs describing single-agent tasks.

7 CONCLUSIONS

In this work, we propose a reward machine (RM) based task representation for cooperative multi-agent reinforcement learning (MARL). The representation allows for a team's task to be decomposed into sub-tasks for individual agents. We accordingly propose a decentralized q-learning algorithm that effectively reduces the MARL problem to a collection of single-agent problems. Experimental results demonstrate the efficiency and scalability of the proposed algorithm, which learns successful team policies even when the baseline algorithms do not converge within the allowed training period. This work demonstrates how well-suited RMs are to the specification and decomposition of MARL problems, and opens interesting directions for future research.

ACKNOWLEDGMENTS

This work was supported in part by ARO W911NF-20-1-0140, DARPA D19AP00004, and ONR N00014-18-1-2829.

REFERENCES

[1] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of model checking*. MIT press.

[2] Filippo Bonchi and Damien Pous. 2013. Checking NFA equivalence with bisimulations up to congruence. *ACM SIGPLAN Notices* 48, 1 (2013), 457–468.

[3] Craig Boutilier. 1996. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th conference on Theoretical aspects of rationality and knowledge*. Morgan Kaufmann Publishers Inc., 195–210.

[4] Alberto Camacho, R Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. 2019. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. 6065–6073.

[5] Christos G Cassandras and Stephane Lafortune. 2009. *Introduction to discrete event systems*. Springer Science & Business Media.

[6] Murat Cubuktepe, Zhe Xu, and Ufuk Topcu. 2020. Policy Synthesis for Factored MDPs with Graph Temporal Logic Specifications. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. 267–275.

[7] Jin Dai and Hai Lin. 2014. Automatic synthesis of cooperative multi-agent systems. In *53rd IEEE Conference on Decision and Control*. IEEE, 6173–6178.

[8] Franck Djeumou, Zhe Xu, and Ufuk Topcu. 2020. Probabilistic Swarm Guidance with Graph Temporal Logic Specifications. In *Proceedings of Robotics: Science and Systems XVI*.

[9] Mohammad Ghavamzadeh, Sridhar Mahadevan, and Rajbala Makar. 2006. Hierarchical multi-agent reinforcement learning. *Autonomous Agents and Multi-Agent Systems* 13, 2 (2006), 197–229.

[10] Carlos Guestrin, Michail Lagoudakis, and Ronald Parr. 2002. Coordinated reinforcement learning. In *International Conference on Machine Learning*, Vol. 2. Citeseer, 227–234.

[11] Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. 2019. A Survey of Learning in Multiagent Environments: Dealing with Non-Stationarity. arXiv:1707.09183 [cs.MA]

[12] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. 2019. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems* 33, 6 (2019), 750–797.

[13] John E Hopcroft. 1971. *A linear algorithm for testing equivalence of finite automata*. Vol. 114. Defense Technical Information Center.

[14] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. 2018. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*. 2112–2121.

[15] Rodrigo Toro Icarte, Ethan Waldie, Toryn Klassen, Rick Valenzano, Margarita Castro, and Sheila McIlraith. 2019. Learning Reward Machines for Partially Observable Reinforcement Learning. In *Advances in Neural Information Processing Systems*. 15497–15508.

[16] Mohammad Karimadini and Hai Lin. 2011. Guaranteed global performance through local coordinations. *Automatica* 47, 5 (2011), 890–898.

[17] Mohammad Karimadini, Hai Lin, and Ali Karimoddini. 2016. Cooperative tasking for deterministic specification automata. *Asian Journal of Control* 18, 6 (2016), 2078–2087.

[18] Ali Karimoddini, Mohammad Karimadini, and Hai Lin. 2014. Decentralized hybrid formation control of unmanned aerial vehicles. In *American Control Conference*. IEEE, 3887–3892.

[19] Jelle R Kok and Nikos Vlassis. 2005. Using the max-plus algorithm for multiagent decision making in coordination graphs. In *Robot Soccer World Cup*. Springer, 1–12.

[20] Ratnesh Kumar and Vijay K Garg. 2012. *Modeling and control of logical discrete event systems*. Vol. 300. Springer Science & Business Media.

[21] Borja G. León and Francesco Belardinelli. 2020. Extended Markov Games to Learn Multiple Tasks in Multi-Agent Reinforcement Learning. arXiv:2002.06000 [cs.AI]

[22] Feng Lin and Walter Murray Wonham. 1988. On observability of discrete-event systems. *Information sciences* 44, 3 (1988), 173–198.

[23] Michael L Littman. 1994. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*. Elsevier, 157–163.

[24] Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. 2019. Maven: Multi-agent variational exploration. In *Advances in Neural Information Processing Systems*. 7613–7624.

[25] Rajbala Makar, Sridhar Mahadevan, and Mohammad Ghavamzadeh. 2001. Hierarchical multi-agent reinforcement learning. In *Proceedings of the 5th International Conference on Autonomous Agents*. 246–253.

[26] Laetitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. 2012. Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. *The Knowledge Engineering Review* 27, 1 (2012), 1–31.

[27] Devaprakash Muniraj, Kyriakos G Vamvoudakis, and Mazen Farhood. 2018. Enforcing signal temporal logic specifications in multi-agent adversarial environments: A deep Q-learning approach. In *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 4141–4146.

[28] Cyrus Neary, Zhe Xu, Bo Wu, and Ufuk Topcu. 2020. Reward Machines for Cooperative Multi-Agent Reinforcement Learning. arXiv:2007.01962 [cs.MA]

[29] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2018. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*. PMLR, 4295–4304.

[30] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. 2019. The StarCraft Multi-Agent Challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. 2186–2188.

[31] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. 2019. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning*. PMLR, 5887–5896.

[32] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. 2018. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems*. 2085–2087.

[33] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[34] Ming Tan. 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the 10th International Conference on Machine Learning*. 330–337.

[35] Hongyao Tang, Jianye Hao, Tangjie Lv, Yingfeng Chen, Zongzhang Zhang, Hangtian Jia, Chunxu Ren, Yan Zheng, Zhaopeng Meng, Changjie Fan, and Li Wang. 2019. Hierarchical Deep Multiagent Reinforcement Learning with Temporal Abstraction. arXiv:1809.09332 [cs.LG]

[36] Arryon D Tijsma, Madalina M Drugan, and Marco A Wiering. 2016. Comparing exploration strategies for q-learning in random stochastic mazes. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 1–8.

[37] Elise Van der Pol and Frans A Oliehoek. 2016. Coordinated deep reinforcement learners for traffic light control. *Proceedings of Learning, Inference and Control of Multi-Agent Systems (at NIPS 2016)* (2016).

[38] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the 13th AAAI Conference on Artificial Intelligence*.

[39] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.

[40] K Wong. 1998. On the complexity of projections of discrete-event systems. In *Proceedings of the International Workshop of Discrete Event Systems*. Citeseer, 201–206.

[41] 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*, Vol. 30. 590–598.

[42] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. 2019. Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms. arXiv:1911.10635 [cs.LG]