Reward Machines for Cooperative Multi-Agent Reinforcement Learning

Cyrus Neary [*]   Zhe Xu [*]   Bo Wu [*]   Ufuk Topcu [* †]

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 and independencies, 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 two orders of magnitude faster than a centralized learner and significantly outperforms hierarchical and independent q-learning approaches.

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.

Two challenges inherent to MARL are coordination and non-stationarity. Firstly, the need for coordination between the agents arises because the correctness of any individual's actions may depend on the actions of its teammates [3, 10]. Secondly, the agents are learning and updating their behaviors simultaneously. Thus 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 provably accomplishes the original 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 learn to 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 a factor of 200 times faster than a centralized learner and performs roughly on par with hierarchical independent learners (h-IL) [30]. In a more complicated temporally extended three-agent task, the proposed algorithm quickly learns effective team behavior while neither h-IL nor independent q-learning (IQL) [29] converge to policies completing the task.

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 distributions over actions.

The goal of reinforcement learning (RL) is to learn an optimal policy π^* maximizing expected future discounted reward from any state [28]. 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 [33], 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 [21, 3]. A team 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 for all $\mathbf{a} \in \mathcal{A}$. A team policy is defined as $\boldsymbol{\pi} : \mathcal{S} \to \Delta(\mathcal{A})$. Analogous to the single agent case, the objective of team MARL is to find a team policy $\boldsymbol{\pi}^*$ that maximizes the expected discounted future reward from any joint state $\mathbf{s} \in \mathcal{S}$.

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 G. However, the red, yellow, and green colored areas are unsafe for agents A_1, A_2, and A_3 respectively; if one of the agents moves into their colored region before the button of the corresponding color has been pressed, the team fails the task. 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 G.

(a) Cooperative buttons domain. (b) Reward machine encoding the cooperative buttons task.
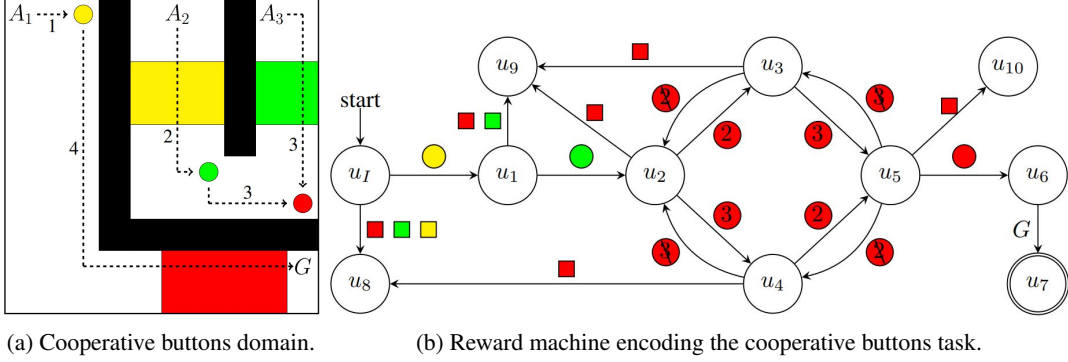
Figure 1: The multi-agent buttons task. In Figure (a), the colored circles denote the locations of the buttons, the thick black areas are walls the agents cannot cross, and the numbered dotted lines show the order of high-level steps necessary to complete the task. The set of events of the RM in (b) is $\Sigma = \{$🔴2, 🔴3̄, 🔴3, 🔴4̄, 🔴, 🟡, 🟢, 🟥, 🟨, 🟩, $G\}$.

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 [14]. 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, 🟥 $\in \Sigma$ corresponds to the event that A_1 has moved into the unsafe red region, and 🔴 $\in \Sigma$ corresponds to the event of the red button being pressed. Because both A_2 and A_3 must simultaneously press the red button for the event 🔴 to occur, we additionally include the events 🔴2, 🔴2̄, 🔴3, 🔴3̄ in Σ, which represent A_2 or A_3 individually either pressing or not pressing the red button.

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, if one of the dangerous colored regions are entered, the corresponding event 🟥, 🟨, or 🟩 will cause a transition to state u_8 from which there are no outgoing transitions; the team has failed the task, so it does not progress from this state. If instead A_1 presses the yellow button, then the event 🟡 will cause the RM to transition to state u_1; A_2 may now safely proceed across the yellow region. 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 [34]; 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 $G \in \Sigma$ occurs while the RM is in state u_6 the task has been successfully completed; the RM will transition to state u_7 and return reward 1.

A run of RM $\mathcal{R}$ on the sequence of events $e_0 e_1 ... e_k \in \Sigma^*$ is a sequence $u_0 e_0 u_1 e_1 ... u_k e_k u_{k+1}$, 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, $\mathcal{R}($🟡🟢🔴2🔴3🔴$G) = 1$, but $\mathcal{R}($🟡🟢🟥$) = 0$.

3.2 Labeling Functions and Q-Learning with Reward Machines

RMs may be applied to the RL setting 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* $L : S \times U \to 2^{\Sigma}$, which abstracts the current environment state to sets of high-level events. Note, however, that L also takes the current RM state

$u \in U$ as input, allowing the events output by L to depend not only on the environment state, but also on the current progress through the task.

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. At time step t, the agent uses the RM state u_t and its estimate of $q_{u_t}(s_t, \cdot)$ to select action a_t. The environment accordingly progresses to state s_{t+1}. The events output by $L(s_{t+1}, u_t)$ then cause the RM to transition to state u_{t+1}, and the corresponding reward output by σ is used to update $q_{u_t}(\mathbf{s}_t, a_t)$ (see supplementary materials §8). The QRM algorithm is guaranteed to converge to an optimal policy.

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.

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 ith 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 ith 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 $\Sigma_1 = \{\blacksquare, \bigcirc, \bullet, G\}$; A_1 should avoid the red region, has access to the yellow button, must know when the red button has been pressed, and should eventually proceed to the goal location. Note that, for example, events $\blacksquare$ and $\square$ are not included in Σ_1 because these regions are not dangerous to A_1, and because A_1 is separated from these colored regions by the walls in the environment. Similarly, event $\bullet$ is not in Σ_1 because for A_1, it is only an intermediate step in the process of pressing the red button. Similarly, the event sets of A_2 and A_3 are $\Sigma_2 = \{\square, \bigcirc, \bullet, ❷, ❹, \bullet\}$ and $\Sigma_3 = \{\blacksquare, \bullet, ❸, ❶, \bullet\}$, respectively.

Extending the definition of natural projections on automata [16] 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. A formal definition of $\mathcal{R}_i$ is provided in the supplementary materials §9. Intuitively, the projection may be constructively defined as follows:

1. Remove all transitions triggered by events *not* contained in Σ_i.
2. To define the projected states U_i, merge all states connected by a removed transition.
3. The remaining transitions are used to define the projected transition function δ_i.
4. The reward states F_i are defined as the collections of merged states containing at least one reward state $u \in F$ from the original RM. Output function σ_i is defined accordingly.

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 $\square$, $\blacksquare$, $\bullet$, ❷, ❹, ❸, and ❶ 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 while avoiding the red region, 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.

Consider some finite event sequence $\xi = e_0...e_k \in \Sigma^*$. The projection of ξ onto Σ_i, denoted $\xi_i \in \Sigma_i^*$, is obtained by removing all events from ξ that are not in Σ_i (see supplementary material §9). ξ_i may be thought of as the event sequence ξ from the point of view of the ith 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
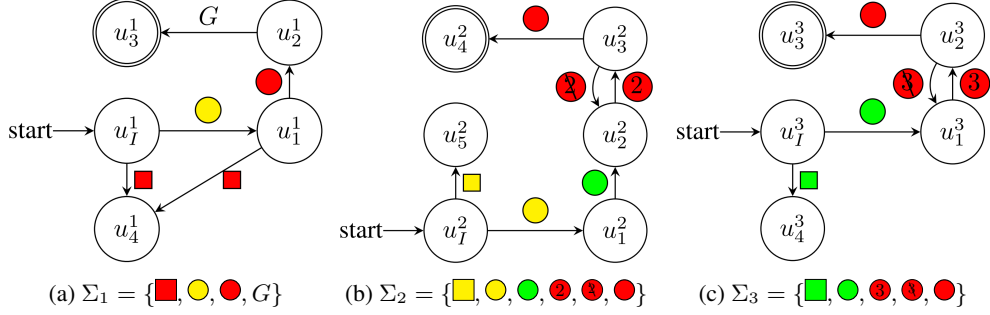
(a) $\Sigma_1 = \{\blacksquare, \bigcirc, \bullet, G\}$ (b) $\Sigma_2 = \{\square, \bigcirc, \bullet, \mathbf{2}, \mathbf{3}, \bullet\}$ (c) $\Sigma_3 = \{\blacksquare, \bullet, \mathbf{3}, \mathbf{3}, \bullet\}$

Figure 2: Projections of the team RM, illustrated in Figure 1b, onto the local event sets Σ_1, Σ_2, Σ_3.

uses the notions of bisimilarity and parallel composition, common concepts for finite transition systems [1], that are formally defined for RMs in supplementary materials §9.

Theorem 1. *Given RM $\mathcal{R}$ and a collection of local event sets Σ_1, Σ_2, ..., Σ_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 $\mathcal{R}_i(\xi_i) = 1$ for all $i = 1, 2, ..., N$. Otherwise, $\mathcal{R}(\xi) = 0$ and $\mathcal{R}_i(\xi_i) = 0$ for all $i = 1, 2, ...N$.*

We note that [16, 17] present conditions, in terms of $\mathcal{R}$ and Σ_1, Σ_2, ... Σ_N, which may be applied to check whether $\mathcal{R}$ is bisimilar to the parallel composition of its projections $\mathcal{R}_1$, $\mathcal{R}_2$, ..., $\mathcal{R}_N$. Alternatively, one may computationally check whether this result holds by constructing the parallel composition of the projected RMs and applying the *Hopcroft-Karp* algorithm to check equivalence with $\mathcal{R}$ [13, 2].

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 ith 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 $L : \mathcal{S} \times U \rightarrow 2^{\Sigma}$ mapping team states $\mathbf{s} \in \mathcal{S}$ and RM states $u \in U$ from $\mathcal{R}$ to sets of events. For example, $L(\mathbf{s}, u) = \{\bullet\}$ 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* $L_i : S_i \times \mathcal{A}_i \rightarrow 2^{\Sigma_i}$ 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 only 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 2. *(Decomposable labeling function) A labeling function $L : \mathcal{S} \times U \rightarrow 2^{\Sigma}$ is considered decomposable with respect to local event sets Σ_1, Σ_2, ..., Σ_N if there exists a collection of local labeling functions L_1, L_2, ..., L_N with $L_i : S_i \times U_i \rightarrow 2^{\Sigma_i}$ such that $L(\mathbf{s}, u)$ outputs event e if and only if $L_i(s_i, u^i)$ outputs event e for every i in I_e, the set of collaborating agents on event e. Here, s_i*

5

is the ith component of the team's joint state **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 constructively define $L_1, ..., L_N$ to satisfy the condition in Definition 2. Following this idea, we 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, here $u \in U$ 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 **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 part of the function's input: $s_i \in S_i$ and $u^i \in U_i$. A formal definition of L_i as well as conditions on L that ensure L_i is well defined are given in supplementary material §10.

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, $\bigcirc \in \Sigma_1 \cap \Sigma_2$ 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 $\bigcirc$, the agents must *syncrhonize* on this event: $\bigcirc$ 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 ith 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 has occurred.

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 $i = 1, 2, ..., N$ (see supplementary material §10).

Theorem 2. *Given $\mathcal{R}$, L, and $\Sigma_1, ..., \Sigma_N$, suppose the bisimilarity condition from Theorem 1 holds. Furthermore, assume L is decomposable with respect to $\Sigma_1, ... \Sigma_N$ 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 $i = 1, 2, ..., N$.*

Let $V^{\boldsymbol{\pi}}(\mathbf{s})$ denote the expected future undiscounted reward 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 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 $\mathcal{R}_i$.

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

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

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 normally be a shared event in the team setting, we simulate a synchronization signal 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 ith 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. For further details, see supplementary material §12.

5   Experimental Results

In this section, we provide empirical evaluations of DQPRM in three task domains. The buttons task is as described in §3. We additionally consider two-agent, and three-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 centralized QRM (CQRM) algorithm described in §3.2, independent q-learners (IQL) [29], and hierarchical independent learners (h-IL) [30]. 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 successfully completed the rendezvous.

Each IQL agent learns a q-function mapping augmented state-action pairs to values. That is, the ith agent learns a q-function $q_i : S_i \times S_M \times A_i \to \mathbb{R}$, where S_i, A_i are the local states and actions of the agent and S_M is the finite set of memory states, commonly observed by the entire team.

Our implementation of h-IL is inspired by the learning structure outlined in [30]. 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 safe (non-colored) region, navigate to the yellow button, and navigate to the goal location.

In all algorithms, we use a discount factor $\gamma = 0.9$, a learning rate $\alpha = 0.8$, and an exploration parameter that is annealed from $\epsilon = 0.3$ to zero as training progresses. 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 5% chance the agent will instead slip to an adjacent state. Each training 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 3 shows the experimental results for each algorithm over 10 separate runs per domain. The figures plot the median number of steps required to complete the team task against the number of

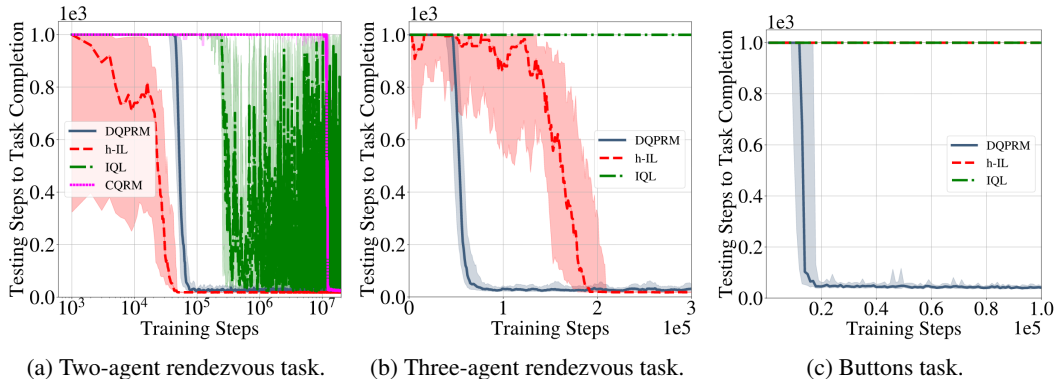(a) Two-agent rendezvous task.   (b) Three-agent rendezvous task.   (c) Buttons task.

Figure 3: Algorithm performance on various tasks. 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. Note that the scale in (a) is logarithmic, whereas it is linear in (b), (c).

elapsed training steps. The shaded regions enclose the 25th and 75th percentiles. We note that the CQRM baseline is only tested in the two-agent scenario because this centralized approach requires excessive amounts of memory for the three-agent tasks; 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 three agents, DQPRM significantly outperforms all baseline methods. The key advantage of the DQPRM algorithm, and what allows it to scale well with task complexity and the number of agents, is that it trains the agents separately using their respective single-agent task projections. This removes the problem of non-stationarity and also allows the agents 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 multi-agent learning in §6.

6   Related Work

Task decomposition in multi-agent systems has been studied from a planning and cooperative control perspective [16, 17, 18, 7]. These works examine the conditions in which group tasks described by automata may be broken into sub-tasks executable by individuals. [8, 6] 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.

The MARL literature is rich [35, 11, 12, 23]. A popular approach to MARL is IQL [29], in which each agent learns independently and treats its teammates as part of the environment. [10, 19, 31, 27, 25, 26] decompose cooperative team tasks by factoring the joint q-function into components. Our work, which examines cooperative tasks that have sparse and temporally delayed rewards, is most closely related to hierarchical approaches to MARL. In particular, [22, 9] 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, [30] 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 [34, 15] demonstrate that, in the single-agent RL setting, RMs can instead 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, [14, 4] demonstrate in the single-agent RL setting that RMs may be applied to continuous environments by replacing tabular q-learning with double deep q networks [32]. We note that 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 for temporally extended cooperative tasks that cannot be solved by the baseline algorithms used for comparison. This work demonstrates how well-suited RMs are to abstraction and decomposition of MARL problems, and opens interesting directions for future research.

References

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

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

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

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

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

[6] Murat Cubuktepe, Zhe Xu, and Ufuk Topcu. Policy synthesis for factored mdps with graph temporal logic specifications. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 267–275, 2020.

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

[8] Franck Djeumou, Zhe Xu, and Ufuk Topcu. Probabilistic swarm guidance with graph temporal logic specifications. In *Proceedings of Robotics: Science and Systems XVI*, 2020.

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

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

[11] Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*, 2017.

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

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

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

[15] Rodrigo Toro Icarte, Ethan Waldie, Toryn Klassen, Rick Valenzano, Margarita Castro, and Sheila McIlraith. Learning reward machines for partially observable reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 15497–15508, 2019.

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

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

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

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

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

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

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

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

[24] Rémi Morin. Decompositions of asynchronous systems. In *International Conference on Concurrency Theory*, pages 549–564. Springer, 1998.

[25] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: monotonic value function factorisation for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1803.11485*, 2018.

[26] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:1905.05408*, 2019.

[27] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. 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*, pages 2085–2087. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

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

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

[30] Hongyao Tang, Jianye Hao, Tangjie Lv, Yingfeng Chen, Zongzhang Zhang, Hangtian Jia, Chunxu Ren, Yan Zheng, Changjie Fan, and Li Wang. Hierarchical deep multiagent reinforcement learning. *arXiv preprint arXiv:1809.09332*, 2018.

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

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

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

[34] Zhe Xu, Ivan Gavran, Yousef Ahmad, Rupak Majumdar, Daniel Neider, Ufuk Topcu, and Bo Wu. Joint inference of reward machines and policies for reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 590–598, 2020.

[35] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *arXiv preprint arXiv:1911.10635*, 2019.

Reward Machines for Cooperative Multi-Agent Reinforcement Learning: Supplementary Material

8 Q-Learning with Reward Machines

Algorithm 1 shows pseudocode for the QRM algorithm, originally introduced in [14]. The algorithm learns a collection $Q = \{q_u | S \times A \to \mathbb{R} | u \in U\}$ of q-functions, each corresponding to a separate policy to follow when in RM state $u \in U$.

At each time step t, the agent takes action a_t and the MDP progresses to the next state s_{t+1}. The agent uses its current RM state to abstract its new environment state through the labeling function, obtaining $L(s_{t+1}, u_t) \in 2^\Sigma$. Each event in $L(s_{t+1}, u_t)$ causes an RM transition and a resulting reward output. Note that for a single environment step, the reward machine may transition any number of times corresponding to the number of events that occurred simultaneously at time t.

Algorithm 1: Q-Learning with Reward Machines

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

1   $Q \leftarrow InitializeQFunctions()$
2   **for** $n = 1$ **to** $NumEpisodes$ **do**
3     $u_1 \leftarrow u_I, s_1 \leftarrow environmentInitialState()$
4   **for** $t = 0$ **to** $NumSteps$ **do**
5       $a \leftarrow getAction(q_{u_1}, s_1)$
6       $s_2 \leftarrow executeAction(s_1, a)$
7       $u_{temp} \leftarrow u_1, r \leftarrow 0$
8   **for** $e \in L(s_2, u_1)$ **do**
9         $u_2 \leftarrow \delta(u_{temp}, e)$
10        $r \leftarrow r + \sigma(u_{temp}, u_2)$
11        $u_{temp} \leftarrow u_2$
12       $q_{u_1}(s_1, a) \leftarrow (1 - \alpha)q_{u_1}(s_1, a) + \alpha(r + \gamma \max_{a' \in A} q_{u_2}(s_2, a'))$
13       $u_1 \leftarrow u_2, s_1 \leftarrow s_2$
14   **if** $u_2 \in F$ **then**
15         $break$
16   **return** Q

---

9 Reward Machine Formalisms

9.1 Reward Machine Transitions on Sequences of Events

Let Σ^* be the set of all finite strings, including the empty string ϵ, over elements from Σ. Any string $\xi = e_0 e_1 ... e_k \in \Sigma^*$ thus represents a *sequence* of environment events. Following the notation of [20], we extend the domain of definition of the transition function δ to $U \times \Sigma^*$.

Definition 3. *(RM transition on a sequence of events) For a reward machine $\mathcal{R}$ and sequence of events $\xi \in \Sigma^*$, we define the transition over sequence ξ from state u recursively with relations $u = \delta(u, \varepsilon)$ and $\delta(u, \xi e) = \delta(\delta(u, \xi), e), \forall \xi \in \Sigma^*, e \in \Sigma$.*

In words, $\delta(u, \xi) \in U$ is the RM state that the sequence of events ξ will cause the RM to transition to, given it begins in state u. Given our definition of the output function σ for a task RM, we note that the RM will return 1 if and only if ξ causes the RM to transition to a final state $u \in F$. i.e. $\mathcal{R}(\xi) = 1$ if and only if $u \notin F$ and $\delta(u, \xi) \in F$, otherwise, $\mathcal{R}(\xi) = 0$.

9.2 Reward Machine Projection

To define the projection of an RM $\mathcal{R}$ onto a local event set $\Sigma_i \subseteq \Sigma$, we begin by defining a notion of state equivalence under local event set Σ_i [24].

11

Definition 4. *(Equivalence class of states) Given a reward machine $\mathcal{R} = (U, u_I, \Sigma, \delta, \sigma, F)$, and a local event set $\Sigma_i \subseteq \Sigma$ we define equivalence relation R_{Σ_i} as the minimal equivalence relation such that for all $u_1, u_2 \in U$, $e \in \Sigma$, if $u_2 = \delta(u_1, e)$ and $e \notin \Sigma_i$, then $(u_1, u_2) \in R_{\Sigma_i}$. The equivalence class of any state $u \in U$ under equivalence relation R_{Σ_i} is $[u]_{\Sigma_i} = \{u' \in U | (u, u') \in R_{\Sigma_i}\}$. The quotient set of U by R_{Σ_i} is defined as the set of all equivalence classes $U/R_{\Sigma_i} = \{[u]_{\Sigma_i} | u \in U\}$.*

In words, two states $u_1, u_2 \in U$ of the RM $\mathcal{R}$ are members of the same equivalence class if a transition (or a sequence of transitions) exists between them that are triggered by events outside of the local event set Σ_i. These equivalence classes represent the collections of states of $\mathcal{R}$ that are indistinguishable to an agent who may only observe events from Σ_i. Now, we use the equivalence classes of states $[u]_{\Sigma_i}$ to define the RM projection onto Σ_i.

Definition 5. *(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/R_{Σ_i}; each state $u^i \in U_i$ is an equivalence class of states from $u \in U$.*
- *The initial state is $u_I^i = [u_I]_{\Sigma_i}$.*
- *Transition function $\delta_i : U_i \times \Sigma_i \to 2^{U_i}$ is defined such that $u_2^i \in \delta_i(u_1^i, e)$ if and only if there exist $u_1, u_2 \in U$ such that $u_1^i = [u_1]_{\Sigma_i}$, $u_2^i = [u_2]_{\Sigma_i}$, 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 $u^i = [u]_{\Sigma_i}\}$.*
- *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.*

We note that δ_i is not necessarily a deterministic transition function. It's possible for δ_i to map some state-event pairs to sets of states instead of to individual states. However, [16] show that $\mathcal{R}$ is bisimilar to the parallel composition of its projections $\mathcal{R}_1, ... \mathcal{R}_N$ (the condition in Theorem 1) if and only if for every $i = 1, 2, ..., N$, $\mathcal{R}_i$ is bisimilar to an RM that *does* have a deterministic transition function. So, in this work we assume that $\mathcal{R}_i$ is always constructed such that δ_i is deterministic. Practically, this may be achieved by merging any states u_2^i and u_3^i if there exists a state u_1^i and an event $e \in \Sigma_i$ such that $\delta_i(u_1^i, e) = \{u_2^i, u_3^i\}$.

Now we formally define the projection of a sequence of events $\xi \in \Sigma^*$ onto a local event set $\Sigma_i \subseteq \Sigma$.

Definition 6. *(Projection of string of events onto local event subset) Given event set Σ and a local event set $\Sigma_i \subseteq \Sigma$, the projection $P_{\Sigma_i} : \Sigma^* \to \Sigma_i^*$ is defined inductively by the relations $P_{\Sigma_i}(\varepsilon) = \varepsilon$ and $\forall \xi \in \Sigma^*$, $\forall e \in \Sigma$:*

$$P_{\Sigma_i}(\xi e) = \begin{cases} P_{\Sigma_i}(\xi)e & \text{if } e \in \Sigma_i \\ P_{\Sigma_i}(\xi) & \text{Otherwise} \end{cases}$$

In definition 6 recall that ε denotes the empty sequence.

9.3 Parallel Composition and Bisimulation

Definition 7. *(Parallel composition of RMs) The parallel composition of two reward machines $\mathcal{R}_i = \langle U_i, u_I^i, \Sigma_i, \delta_i, \sigma_i, F_i \rangle$, $i = 1, 2$ is defined as $\mathcal{R}_1 \parallel \mathcal{R}_2 = \langle U, u_I, \Sigma, \delta, \sigma, F \rangle$ where*

- *The set of states is defined as $U = U_1 \times U_2$.*
- *The initial state is $u_I = (u_I^1, u_I^2)$.*
- *The set of events is defined as $\Sigma = \Sigma_1 \cup \Sigma_2$.*
- *δ is defined such that for every $(u_1, u_2) \in U_1 \times U_2$ and for every event $e \in \Sigma$,*

$$\delta((u_1, u_2), e) = \begin{cases} (\delta_1(u_1, e), \delta(u_2, e)), & \text{if } \delta_1(u_1, e), \ \delta_2(u_2, e) \text{ defined}, \ e \in \Sigma_1 \cap \Sigma_2 \\ (\delta_1(u_1, e), u_2), & \text{if } \delta_1(u_1, e) \text{ defined}, \ e \in \Sigma_1 \setminus \Sigma_2 \\ (u_1, \delta_2(u_2, e)), & \text{if } \delta_2(u_2, e) \text{ defined}, \ e \in \Sigma_2 \setminus \Sigma_1 \\ \text{undefined}, & \text{otherwise} \end{cases}$$

- *The set of final states is defined as $F = F_1 \times F_2$*
- *The output function $\sigma : U \times U \to \mathbb{R}$ is defined such that $\sigma(u, u') = 1$ if $u_1 \notin F$, $u_2 \in F$ and $\sigma = 0$ otherwise.*

By $\parallel_{i=1}^N \mathcal{R}_i$ we denote the parallel composition of a collection of RMs $\mathcal{R}_1, \mathcal{R}_2, ..., \mathcal{R}_N$. The parallel composition of more than two RMs is defined using the associative property of the parallel composition operator $\parallel_{i=1}^N \mathcal{R}_i = \mathcal{R}_1 \parallel (\mathcal{R}_2 \parallel (... \parallel (\mathcal{R}_{N-1} \parallel \mathcal{R}_N)))$ [5].

Definition 8. *(Bisimilarity of Reward Machines) Let $\mathcal{R}_i = \langle U_i, u_I^i, \Sigma, \delta_i, \sigma_i, F_i \rangle$, $i = 1, 2$, be two RMs. $\mathcal{R}_1$ and $\mathcal{R}_2$ are bisimilar, denoted $\mathcal{R}_1 \cong \mathcal{R}_2$ if there exists a relation $R \subseteq U_1 \times U_2$ with respect to common event set Σ such that*

1. *$(u_I^1, u_I^2) \in R$.*
2. *For every $(u^1, u^2) \in R$,*
   - *$u^1 \in F_1$ if and only if $u^2 \in F_2$*
   - *if $u^{1\prime} \in \delta_1(u^1, e)$ for some $e \in \Sigma$, then there exists $u^{2\prime} \in U_2$ such that $u^{2\prime} \in \delta_2(u^2, e)$ and $(u^{1\prime}, u^{2\prime}) \in R$.*
   - *if $u^{2\prime} \in \delta_2(u^2, e)$ for some $e \in \Sigma$, then there exists $u^{1\prime} \in U_1$ such that $u^{1\prime} \in \delta_1(u^1, e)$ and $(u^{1\prime}, u^{2\prime}) \in R$.*

10 Local Labeling Functions

10.1 Constructive Definition of Local Labeling Function

Definition 9. *(Consistent set of team states) Given agent i's local state $s_i \in S_i$ and the state $u^i \in U_i$ of its projected RM, we define the set $B_{s_i, u^i} \subseteq \mathcal{S} \times U$ as*

$$B_{s_i, u^i} = \{(\mathbf{s}, u) | \mathbf{s} \in S_1 \times S_2 \times ... \times \{s_i\} \times ... \times S_N, \ u \in u_i\}$$

For clarity, recall that any state u^i in the projected RM $\mathcal{R}_i$ corresponds to a collection of states from U. Thus any pair $(\mathbf{s}, u) \in B_{s_i, u^i}$ corresponds to a team environment state $\mathbf{s} \in \mathcal{S}$ such that agent i is in local state s_i and to a team RM state $u \in U$ belonging to the collection $u_i \in U_i$. In words, B_{s_i, u_i} is the set of all pairs of team environment states and team RM states that are consistent with the local environment state s_i and projected RM state u_i.

Definition 10. *(Local labeling function) Given a team labeling function L and a local event set Σ_i, we define the local labeling function L_i, for all $(s_i, u_i) \in S_i \times U_i$, as*

$$L_i(s_i, u_i) = \begin{cases} L(\mathbf{s}, u) \cap \Sigma_i, & \text{if } \exists (\mathbf{s}, u) \in B_{s_i, u_i} \text{ such that } L(\mathbf{s}, u) \cap \Sigma_i \neq \emptyset \\ \emptyset, & \text{if } \forall (\mathbf{s}, u) \in B_{s_i, u_i}, \ L(\mathbf{s}, u) \cap \Sigma_i = \emptyset. \end{cases}$$

We define the following three conditions to ensure that $L_i : S_i \times U_i \to 2^{\Sigma_i}$ is well defined, and that the collection $L_1, ..., L_N$ can be used in place of L. These conditions must hold for $i = 1, 2, ..., N$.

1. To ensure L_i maps to singleton sets of local events, or to the empty set, we must have that for any $(\mathbf{s}, u) \in \mathcal{S} \times U$, $|L(\mathbf{s}, u) \cap \Sigma_i| \leq 1$.
2. Given any input (s_i, u_i), to ensure that $L_i(s_i, u_i)$ has a unique output, there must exist a unique $e \in \Sigma_i$ such that $L(\mathbf{s}, u) \cap \Sigma_i = \{e\}$ or $L(\mathbf{s}, u) \cap \Sigma_i = \emptyset$ for every $(\mathbf{s}, u) \in B_{s_i, u_i}$.
3. For every event $e \in \Sigma$, if $e \notin L(\mathbf{s}, u)$ then there must exists some local event set Σ_i containing e such that $L(\tilde{\mathbf{s}}, \tilde{u}) \cap \Sigma_i = \emptyset$ for every $(\tilde{\mathbf{s}}, \tilde{u}) \in B_{s_i, u_i}$. Here s_i is the local state of agent i consistent with team state $\mathbf{s}$, and u_i is the state of $\mathcal{R}_i$ containing u.

The first condition ensures that L only returns a single event from the local event set of each agent for any step of the environment. The second condition ensures that given the local state pair (s_i, u_i) of agent i, the same event singleton $\{e\} \in 2^{\Sigma_i}$ is returned by L_i regardless of the states of the teammates of agent i. The final condition ensures that e is returned by L only if it is also returned by L_i for every $i \in I_e$.

10.2 Labeling Trajectories of Environment States

For any finite trajectory $\mathbf{s}_0 \mathbf{s}_1 ... \mathbf{s}_k$ of team environment states, we may use the reward machine $\mathcal{R}$ and labeling function L to define a sequence of triplets $(\mathbf{s}_0, u_0, l_0)(\mathbf{s}_1, u_1, l_1) ... (\mathbf{s}_k, u_k, l_k)$ and a string of events $L(\mathbf{s}_0 ... \mathbf{s}_k) \in \Sigma^*$. Here u_t is the state of team RM $\mathcal{R}$ at time t and l_t is the output of the labeling function L at time t. Algorithm 2 details the constructive definition of the sequence.

Similarly, for a collection of local environment states $\{s_0^i s_1^i ... s_k^i\}_{i=1}^N$ and the collections of projected reward machines $\{\mathcal{R}_i\}_{i=1}^N$ and local labeling functions $\{L_i\}_{i=1}^N$ we may define the collection of sequences $\{(s_0^i, u_0^i, \tilde{l}_0^i)(s_1^i, u_1^i, \tilde{l}_1^i) ... (s_k^i, u_k^i, \tilde{l}_k^i)\}_{i=1}^N$ as well as a collection of strings of events

Algorithm 2: Construct sequence of RM states and labeling function outputs.

Input: $\mathbf{s}_0\mathbf{s}_1...\mathbf{s}_k, \mathcal{R}, L$
Output: $(\mathbf{s}_0, u_0, l_0)(\mathbf{s}_1, u_1, l_1)...(\mathbf{s}_k, u_k, l_k), L(\mathbf{s}_0\mathbf{s}_1...\mathbf{s}_k)$

1   $u_0 \leftarrow u_I, l_0 \leftarrow \emptyset, \xi \leftarrow emptyString()$
2   **for** $t = 1$ **to** $k - 1$ **do**
3      $u_{temp} \leftarrow u_t$
4   **for** $e \in l_t$ **do**
5         $u_{temp} \leftarrow \delta(u_{temp}, e)$
6         $\xi \leftarrow append(\xi, e)$
7      $u_{t+1} \leftarrow u_{temp}$
8      $l_{t+1} \leftarrow L(\mathbf{s}_{t+1}, u_t)$
9   $L(\mathbf{s}_0\mathbf{s}_1...\mathbf{s}_k) \leftarrow \xi$
10 **return** $(\mathbf{s}_0, u_0, l_0)(\mathbf{s}_1, u_1, l_1)...(\mathbf{s}_k, u_k, l_k), L(\mathbf{s}_0\mathbf{s}_1...\mathbf{s}_k)$

$\{L_i(s_0^i...s_k^i)\}_{i=1}^N$. Here, u_t^i is the state of projected RM $\mathcal{R}_i$ and $\tilde{l}_t^i$ is the output of local labeling function L_i at time t, after synchronization with collaborating teammates on shared events. Algorithm 3 details the constructive definition of the sequence.

Algorithm 3: Construct sequence of projected RM states and synchronized labeling function outputs.

Input: $\{s_0^i s_1^i...s_k^i\}_{i=1}^N, \{\mathcal{R}_i\}_{i=1}^N, \{L_i\}_{i=1}^N$
Output: $\{(s_0^i, u_0^i, \tilde{l}_0^i)(s_1^i, u_1^i, \tilde{l}_1)...(s_k^i, u_k^i, \tilde{l}_k^i)\}_{i=1}^N, \{L_i(s_0^i s_1^i...s_k^i))\}_{i=1}^N$

1   **for** $i = 1$ **to** N **do**
2      $u_0^i \leftarrow u_I^i, \tilde{l}_0^i \leftarrow \emptyset, \xi_i \leftarrow emptyString()$
3   **for** $t = 1$ **to** $k - 1$ **do**
4   **for** $i = 1$ **to** N **do**
5         $u_{temp}^i \leftarrow u_t^i$
6   **for** $e \in \tilde{l}_t^i$ **do**
7            $u_{temp}^i \leftarrow \delta_i(u_{temp}^i, e)$
8            $\xi_i \leftarrow append(\xi_i, e)$
9         $u_{t+1}^i \leftarrow u_{temp}^i$
10        $l_{t+1}^i \leftarrow L_i(s_{t+1}^i, u_t^i)$
11   **for** $i = 1$ **to** N **do**
12         $I_e \leftarrow getCollaboratingAgents(l_{t+1}^i)$
13         $\tilde{l}_{t+1}^i \leftarrow \bigcap_{j \in I_e} l_{t+1}^j$, (Synchronization step)
14 **for** $i = 1$ **to** N **do**
15      $L_i(s_0^i s_1^i...s_k^i)) \leftarrow \xi_i$
16 **return** $\{(s_0^i, u_0^i, \tilde{l}_0^i)(s_1^i, u_1^i, \tilde{l}_1)...(s_k^i, u_k^i, \tilde{l}_k^i)\}_{i=1}^N, \{L_i(s_0^i s_1^i...s_k^i))\}_{i=1}^N$

We note that the sequences $l_0 l_1...l_k \in (2^\Sigma)^*$ and $\tilde{l}_0^i \tilde{l}_1^i...\tilde{l}_k^i \in (2^{\Sigma_i})^*$ of labeling function outputs are sequences of *sets* of events. However, given our assumption that $L_i(s_{t+1}^i, u_t^i)$ outputs at most one event per time step, $|\tilde{l}_t^i| \leq 1$ for every t. This assumption corresponds to the idea that only one event may occur to an individual agent per time step.

However, the team labeling function L may return multiple events at a given time step, corresponding to all the events that occur concurrently to separate agents. Because $\mathcal{R}$ is assumed to be equivalent to the parallel composition of a collection of component RMs, all such concurrent events are interleaved [1]; the order in which they cause transitions in $\mathcal{R}$ doesn't matter. So, Given $l_1 l_2...l_k \in (2^\Sigma)^*$, the corresponding sequence of events $L(\mathbf{s}_0\mathbf{s}_1...\mathbf{s}_k) \in \Sigma^*$ is constructed by iteratively appending elements in l_t to $L(\mathbf{s}_0\mathbf{s}_1...\mathbf{s}_k)$, as detailed in Algorithm 3. Similarly, from sequence $\tilde{l}_1^i \tilde{l}_2^i...\tilde{l}_k^i \in (2^{\Sigma_i})^*$ we construct the sequence of local events $L_i(s_0^i s_1^i...s_k^i) \in \Sigma_i^*$.

11 Theorems

Theorem 1. *Given RM $\mathcal{R}$ and a collection of local event sets Σ_1, Σ_2, ..., Σ_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 $\mathcal{R}_i(\xi_i) = 1$ for all $i = 1, 2, ..., N$. Otherwise, $\mathcal{R}(\xi) = 0$ and $\mathcal{R}_i(\xi_i) = 0$ for all $i = 1, 2, ...N$.*

Proof. For clarity, recall that $\xi_i \in \Sigma_i^*$ denotes the projection $P_{\Sigma_i}(\xi)$.

Let $\mathcal{R}_p = \langle U_p, u_I^p, \Sigma, \delta_p, \sigma_p, F_p \rangle = \|_{i=1}^N \mathcal{R}_i$ 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, $\mathcal{R}_i(\xi_i) = 1$ if and only if $\delta_i(u_I^i, \xi_i) \in F_i$ for every $i = 1, ..., N$. So, it is sufficient to show that $\delta(u_I, \xi) \in F$ if and only if $\delta_i(u_I^i, \xi_i) \in F_i$ for every $i = 1, ..., N$. Given the assumption that $\mathcal{R} \cong \mathcal{R}_p$, 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 $\delta_i(u_I^i, \xi_i) \in F_i$ for every $i = 1, ..., N$ [20], concluding the proof. $\square$

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 $P_{\Sigma_i}(L(\mathbf{s}_0\mathbf{s}_1...\mathbf{s}_k)) = L_i(s_0^i s_1^i...s_k^i)$ holds. Recall that $L(\mathbf{s}_0\mathbf{s}_1...\mathbf{s}_k) \in \Sigma^*$ denotes the sequence of events resulting from team trajectory $\mathbf{s}_0...\mathbf{s}_k$, as described in §10. $P_{\Sigma_i}(L(\mathbf{s}_0\mathbf{s}_1...\mathbf{s}_k))$ denotes the projection of this sequence onto local event set Σ_i, as defined in §9. Finally, $L_i(s_0^i s_1^i...s_k^i) \in \Sigma_i^*$ denotes the sequence of events output by local labeling function L_i, assuming the agent synchronizes with its teammates on shared events.

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 use induction to show that the appropriate notion of equivalence holds at each time step $t = 1, ..., k$. We write this statement as $l_t \cap \Sigma_i = \tilde{l}_t^i$ for every $i = 1, ..., N$ and every $t = 1, ..., k$. Here, l_t is the output of labeling function L at time t and $\tilde{l}_t^i$ is the synchronized output of local labeling function L_i (§10.2).

At time $t = 0$, l_0 and $\tilde{l}_0^i$ are defined to be empty sets: no events have yet occurred (§10.2). So, trivially $l_0 \cap \Sigma_i = \tilde{l}_0^i$. Furthermore, $u_I \in u_I^i$ by definition of the projected initial state u_I^i. Recall that $u_I \in U$ is the initial state of RM $\mathcal{R}$, and $u_I^i \in U_i$ is the initial state of projected RM $\mathcal{R}_i$.

Now suppose that at some arbitrary time t, $l_t \cap \Sigma_i = \tilde{l}_t^i$ and $u_t \in u_t^i$ for every $i = 1, ..., N$. We wish to show that this implies $l_{t+1} \cap \Sigma_i = \tilde{l}_{t+1}^i$ and $u_{t+1} \in u_{t+1}^i$.

Showing $l_{t+1} \cap \Sigma_i = \tilde{l}_{t+1}^i$: Recall our assumption that for any pair (s_{t+1}^i, u_t^i), $L_i(s_{t+1}^i, u_t^i)$ outputs only one event, corresponding to the idea that only one event may occur to an individual agent per time step. Thus for any $i = 1, ..., N$, $l_{t+1} \cap \Sigma_i = L(\mathbf{s}_{t+1}, u_t) \cap \Sigma_i$ is either equal to $\{e\}$ for some $e \in \Sigma_i$ or it is equal to the empty set.

- If $L(\mathbf{s}_{t+1}, u_t) \cap \Sigma_i = \{e\}$, then $L_j(s_{t+1}^j, u_t^j) = \{e\}$ for every $j \in I_e$ by definition of L being decomposable with corresponding local labeling functions $L_1, L_2, ..., L_N$. Thus $\tilde{l}_{t+1}^i = \bigcap_{j \in I_e} L_j(s_{t+1}^j, u_t^j) = \{e\}$. Recall that $I_e = \{i | e \in \Sigma_i\}$.
- Suppose instead that $L(\mathbf{s}_{t+1}, u_t) \cap \Sigma_i = \emptyset$.
    - If $L_i(s_{t+1}^i, u_t^i) = \emptyset$, then clearly $\tilde{l}_{t+1}^i = \emptyset$.
    - If $L_i(s_{t+1}^i, u_t^i) = \{e\}$ for some $e \in \Sigma_i$, then there exists some $j \in I_e$ such that $e \notin L_j(s_{t+1}^j, u_t^j)$ by the definition of L being decomposable. Thus $\tilde{l}_{t+1}^i = \emptyset$.

<u>Showing $u_{t+1} \in u_t^i$:</u> We begin by using the knowledge that $l_t \cap \Sigma_i = \tilde{l}_{t+1}^i$, and we again proceed by considering the two possible cases.

- If $l_t \cap \Sigma_i = \tilde{l}_t^i = \emptyset$, then the projected RM $\mathcal{R}_i$ will not undergo a transition and so $u_{t+1}^i = u_t^i$. We know that $u_t \in u_t^i$, and that RM $\mathcal{R}$ doesn't undergo any transition triggered by an event in Σ_i. So, by definition the projected states of $\mathcal{R}_i$, we have $u_{t+1} \in u_t^i$. Thus, $u_{t+1} \in u_{t+1}^i$.

- Now consider the case $l_t \cap \Sigma_i = \tilde{l}_t^i = \{e\}$. Projected RM $\mathcal{R}_i$ will transition to a new state u_{t+1}^i according to $\delta_i(u_t^i, e)$. Assume, without loss of generality, that l_t contains events outside of Σ_i which trigger transitions both before and after the transition triggered by e. That is, suppose $a, b \in l_t \setminus \Sigma_i$ and that $\mathcal{R}$ undergoes the following sequence of transitions: $u' = \delta(u_t, a)$, $\tilde{u} = \delta(u', e)$, and finally $u_{t+1} = \delta(\tilde{u}, b)$. Because $u_t \in u_t^i$ and $a \notin \Sigma_i$, we know $u' \in u_t^i$. Because $e \in \Sigma_i$, $\tilde{u} \in \tilde{u}^i$ for some $\tilde{u}^i \in U_i$ not necessarily equal to u_t^i. Finally, because $b \notin \Sigma_i$, we have $u_{t+1} \in \tilde{u}^i$. So, $u_{t+1} \in \tilde{u}^i$ for some projected state $\tilde{u}^i$ such that there exist states $u' \in u_t^i$ and $\tilde{u} \in \tilde{u}^i$ such that $\tilde{u} = \delta(u', e)$ where $e \in \Sigma_i$. By our definition of δ_i and our enforcement of it being a deterministic transition function (§9), u_{t+1}^i is the unique such state in U_i, which implies $\tilde{u}^i = u_{t+1}^i$. Thus $u_{t+1} \in u_{t+1}^i$.

By induction we conclude the proof.

$\square$

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

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

Proof. The Fréchet conjunction inequality states that if $E_1, E_2, ..., E_N$ are a collection of events with probabilities $P(E_1), P(E_2), ..., P(E_N)$, then:

$$\max\{0, P(E_1) + P(E_2) + ... + P(E_N) - (N-1)\} \le P(E_1 \wedge E_2 \wedge ... \wedge E_N)$$
$$\le \min\{P(E_1), P(E_2), ..., P(E_N)\}.$$

Recall that $V^{\boldsymbol{\pi}}(\mathbf{s}_0)$ denotes the expected future undiscounted reward returned by $\mathcal{R}$, given the team follows joint policy $\boldsymbol{\pi}$ from joint environment state $\mathbf{s}_0 \in \mathcal{S}$. We begin by noting that $V^{\boldsymbol{\pi}}(\mathbf{s}_0)$ is equivalent to the probability that the task encoded by $\mathcal{R}$ is eventually completed, given the team follows policy $\boldsymbol{\pi}$ from the initial state $\mathbf{s}_0$ and $\mathcal{R}$ begins in state u_I. To see this, we note that given an initial team state $\mathbf{s}_0 \in \mathcal{S}$, any (possibly history dependent) team policy induces a probability distribution over the set of all possible trajectories $\mathbf{s}_0 \mathbf{s}_1 ...$ of team states. For any such trajectory, the corresponding sum of undiscounted rewards returned by $\mathcal{R}$ will be 1 if the trajectory results in the completion of the task described by $\mathcal{R}$, and it will be 0 otherwise. So, $V^{\boldsymbol{\pi}}(\mathbf{s}_0)$ is the expected value of a Bernoulli random variable which takes the value 1 if and only if the task is completed, and thus is equivalent to the probability of the task being completed under policy $\boldsymbol{\pi}$. Similarly, $V_i^{\boldsymbol{\pi}}(\mathbf{s}_0)$ which is defined as the expected undiscounted future reward returned by $\mathcal{R}_i$, is equivalent to the probability of the task encoded by $\mathcal{R}_i$ being completed given the team follows policy $\boldsymbol{\pi}$.

We note that trajectory $\mathbf{s}_0 \mathbf{s}_1 ...$ will result in the eventual completion of the task described by $\mathcal{R}$ if and only if there exists some $k > 0$ such that $\mathcal{R}(L(\mathbf{s}_0, ... \mathbf{s}_k)) = 1$. By the result of Theorem 2 however, $\mathcal{R}(L(\mathbf{s}_0, ... \mathbf{s}_k)) = 1$ if and only if $\mathcal{R}_i(L_i(s_0^i ... s_k^i)) = 1$ for all $i = 1, ..., N$. So, the trajectory will result in the eventual completion of the task described by $\mathcal{R}$ if and only if it also results in the eventual completion of all projected tasks described by $\mathcal{R}_i$, $i = 1, ..., N$. So, the likelihood of completing the task encoded by $\mathcal{R}$ under policy $\boldsymbol{\pi}$ is equivalent to the likelihood of completing all the tasks encoded by the collection $\{\mathcal{R}_i\}_{i=1}^N$ under the same policy. Thus, with 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.

$\square$

12 DQPRM Execution Details

Figure 4 details the interaction between the agents and the environment for a single time step during testing. The ith agent uses its state s_i, its local RM state u^i, and its learned collection of q-functions Q_i to select an action a_i. Action a_i contributes to the team's joint action $\mathbf{a} = (a_1, ..., a_N)$, which induces an environment transition to a new joint state $\mathbf{s}' = (s'_1, ..., s'_N)$. Each agent then interprets its local state pair (s'_i, u^i) through its local labeling function L_i to obtain the event $e \in \Sigma_i$ occurring at that time step. If e is a shared event, agent i will synchronize with all collaborating agents before updating the state of its local RM $\mathcal{R}_i$. The team is successful if it is rewarded by the team RM $\mathcal{R}$.
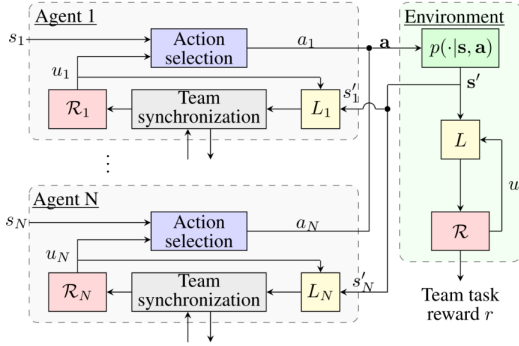

Figure 4: Flowchart of DQPRM policy execution.

13 A Ten-Agent Experiment

Figure 5 shows the experimental results for a ten-agent variant of the rendezvous task described in §5. To successfully complete the task, all agents must simultaneously occupy the rendezvous location before proceeding to their respective goal locations. We observe that while the DQPRM algorithm converges to a team policy relatively quickly, the baseline methods fail to converge within the allowed number of training steps. For comparison, we recall that in the two-agent variant of the same task, h-IL outperforms DQPRM. In the three-agent variant of the task, DPQRM outperforms h-IL but h-IL converges within the allowed $3e5$ training steps. This demonstrates the ability of the proposed DQPRM algorithm to scale relatively well with the number of agents. This capability is owed to the fact that by decomposing the task, DQPRM is able to train each agent entirely separately from its teammates, while ensuring that the resulting composite behavior accomplishes for team's task.
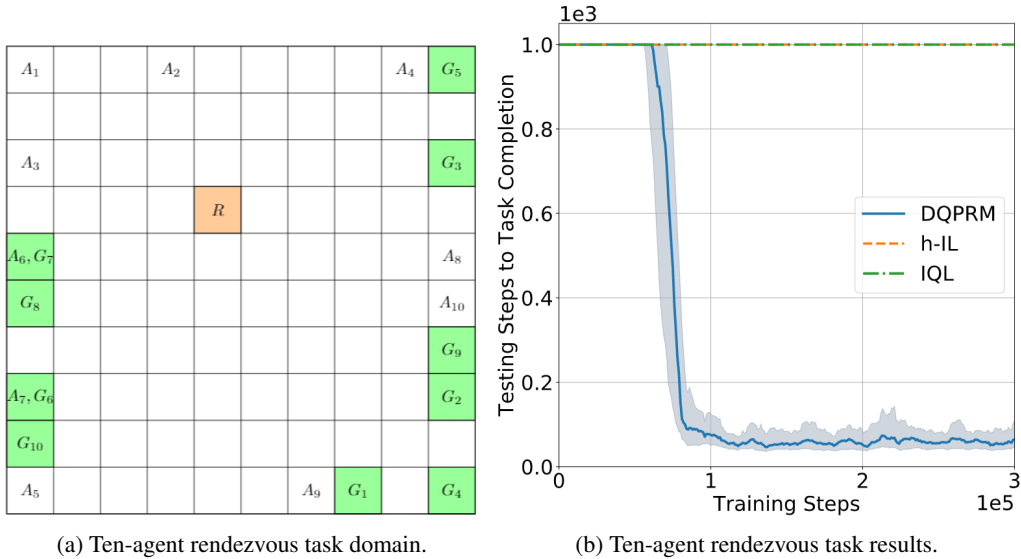
(a) Ten-agent rendezvous task domain.
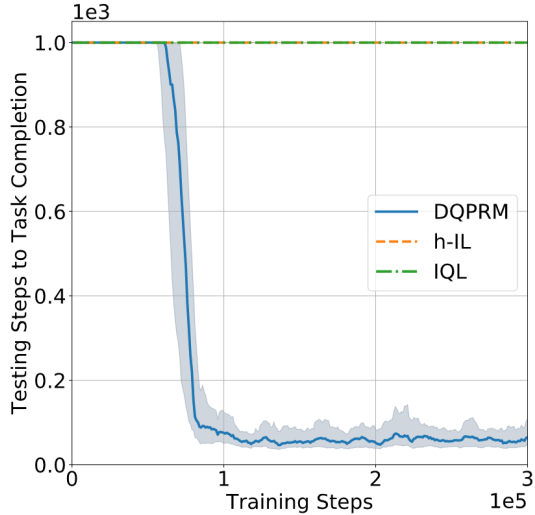
(b) Ten-agent rendezvous task results.

Figure 5: (a) The initial position of agent i is marked A_i. The common rendezvous location for all agents is marked R and is highlighted in orange. The goal location for agent i is marked G_i and highlighted in green. (b) Algorithm performance on the ten-agent rendezvous task. The y-axis shows the number of steps required for the learned policies to complete the task. The x-axis shows the number of elapsed training steps.