DistSPECTRL: Distributing Specifications in Multi-Agent Reinforcement Learning Systems

Joe Eappen
Purdue University
jeappen@purdue.edu

Suresh Jagannathan
Purdue University
suresh@cs.purdue.edu

ABSTRACT

While notable progress has been made in specifying and learning objectives for general cyber-physical systems, applying these methods to distributed multi-agent systems still pose significant challenges. Among these are the need to (a) craft specification primitives that allow expression and interplay of both local and global objectives, (b) tame explosion in the state and action spaces to enable effective learning, and (c) minimize coordination frequency and the set of engaged participants for global objectives. To address these challenges, we propose a novel specification framework that allows natural composition of local and global objectives used to guide training of a multi-agent system. Our technique enables learning expressive policies that allow agents to operate in a coordination-free manner for local objectives, while using a decentralized communication protocol for enforcing global ones. Experimental results support our claim that sophisticated multi-agent distributed planning problems can be effectively realized using specification-guided learning.

KEYWORDS

Multi-Agent Reinforcement Learning, Specification-Guided Learning, Reward Shaping

1 INTRODUCTION

Reinforcement Learning (RL) can be used to learn complex behaviors in many different problem settings. A main component of RL is providing feedback to an agent via a reward signal. This signal should encourage desired behaviors, and penalize undesirable ones, enabling the agent to eventually proceed through a sequence of tasks and is designed by the programmer beforehand. A commonly used technique to encode tasks in a reward signal is the sparse method of providing zero reward until a task is completed upon which a non-zero reward is given to the agent. Because this procedure has the significant shortcoming of delaying generating a useful feedback signal for a large portion of the agent-environment interaction process, a number of alternative techniques have been proposed [1, 16].

Formulating a reward signal that reduces the sparsity of this feedback is known as *reward shaping*. Often, this is done manually but a more general, robust method would be to automatically shape a reward given a specification of desired behavior. SPECTRL [6] proposes a reward shaping mechanism for a set of temporal logic specifications on a single-agent task that uses a compiled a finite-state automaton called a task monitor. Reward machines [2, 20, 21] are another objective-specifying method for RL problems that also define a finite automaton akin to the ones used in SPECTRL, with some subtle differences such as the lack of registers (used by the task monitor for memory).

Distributed multi-agent applications however, introduce new challenges in automating this reward shaping process. Agents have their own respective goals to fulfill as well as coordinated goals that must be performed in cooperation with other agents. While the expressiveness of the language in SPECTRL lends itself, with minor extensions, to specifying these kinds of goals, we require new compilation and execution algorithms to tackle inherent difficulties in multi-agent reinforcement learning (MARL); these include credit assignment of global objectives and the presence of large state and action spaces that grow as the number of agents increases.

Learning algorithms for multi-agent problems have often encouraged distribution as a means of scaling in the presence of state and action space explosion. This is because purely centralized approaches have the disadvantages of not only requiring global knowledge of the system at all times but also induce frequent and costly synchronized agent control.

To address these issues, we develop a new specification-guided distributed multi-agent reinforcement learning framework. Our approach has four main features. First, we introduce two classes of predicates (*viz.* local and global) to capture tasks in a multi-agent world (Sec. 4). Second, we develop a new procedure for generating composite task monitors using these predicates and devise new techniques to distribute these monitors over all agents to address scalability and decentralization concerns (Sec. 5). Third, we efficiently solve the introduced problem of subtask synchronization (Sec. 6) among agents via synchronization states in the task monitors. Lastly, we describe a wide class of specification structures (Sec. 7) amenable to scaling in the number of agents and provide a means to perform such a scaling (Sec. 8).

By using these components in tandem, we provide the first solution to composing specifications and distributing them among agents in a scalable fashion within a multi-agent learning scenario supporting continuous state and action spaces. Before presenting details of our approach, we first provide necessary background information (Sec. 2) and formalize the problem (Sec. 3).

2 BACKGROUND

Markov Decision Processes. Reinforcement learning is a tool to solve Markov Decision Processes (MDPs). MDPs are tuples of the form $\langle \mathcal{S}, D, A, P, R, T \rangle$ where $\mathcal{S} \in \mathbb{R}^n$ is the state space, D is the initial state distribution, $A \in \mathbb{R}^m$ is the action space, $P : \mathcal{S} \times A \times \mathcal{S} \to [0, 1]$ is the transition function, and T is the time horizon. A rollout $\zeta \in Z$ of length T is a sequence of state and action pairs $\zeta = (s_0, a_0, ..., a_{T-1}, s_T)$ where $s_i \in \mathcal{S}$ and $a_i \in A$ are such that $s_{i+1} \sim P(s_i, a_i)$. $R : Z \to \mathbb{R}$ here is a reward function used to score a rollout ζ.

Multi-Agent Reinforcement Learning. A Markov game is a tuple $\mathcal{M}_g = \langle \mathcal{N}, \{\mathcal{S}^i\}_{i\in\mathcal{N}}, D, \{A^i\}_{i\in\mathcal{N}}, P, \{R^i\}_{i\in\mathcal{N}}, T \rangle$ where $\mathcal{N} =$

Proc. of the Adaptive and Learning Agents Workshop (ALA 2022), Cruz, Hayes, da Silva, Santos (eds.), May 9-10, 2022, Online, https://ala2022.github.io/. 2022.

$\{1, \cdots, N\}$ denotes the set of N agents and A^i, R^i define their agent-specific action spaces and reward functions. They are a direct generalization of MDPs to the multi-agent scenario. Let $\mathcal{S}_m = \{\mathcal{S}^i\}_{i \in \mathcal{N}}$ and $A_m = \{A^i\}_{i \in \mathcal{N}}$, then $P : \mathcal{S}_m \times A_m \times \mathcal{S}_m \rightarrow [0, 1]$ is the transition function. A rollout $\zeta_m \in Z_m$ here corresponds to $\zeta_m = (\bar{s}_0, \bar{a}_0, ..., \bar{a}_{T-1}, \bar{s}_T)$ where $\bar{s} \in \mathcal{S}_m$ and $\bar{a} \in A_m$. We also define an agent specific rollout $\zeta^i_m \in Z^i_m$, $\zeta^i_m = (s^i_0, a^i_0, ..., a^i_{T-1}, s^i_T)$ where $s^i \in \mathcal{S}^i$ and $a^i \in A^i$. D is the initial state distribution over $\mathcal{S}_m$.

Agents attempt to learn a policy $\pi^i : \mathcal{S}^i \rightarrow \Delta(A^i)$ such that $\mathbb{E}\left[\sum_t R^i_t | \pi^i, \pi^{-i}\right]$ is maximized, where $\Delta(A^i)$ is a probability distribution over A^i and π^{-i} is the set of all policies apart from π^i. We use $\Pi = \{\pi^i\}_{i \in \mathcal{N}}$ to denote the set of all agent policies.

For simplicity, we restrict our formulation to a *homogeneous* set of agents which operate over the same state ($\mathcal{S}^i = \mathcal{S}_A$) and action space ($A^i = A_A$).

SPECTRL. Jothimurugan *et. al* [6] introduce a specification language for reinforcement learning problems built using temporal logic constraints and predicates. It is shown to be adept at handling complex compositions of task specifications through the use of a *task monitor* and well-defined monitor transition rules. Notably, one can encode Non-Markovian tasks into the MDP using the additional states of the automaton (task monitor) compiled from the given specification.

The atomic elements of this language are Boolean predicates b defined as functions of a state $\mathcal{S}$ with output $[\![b]\!] : \mathcal{S} \rightarrow \mathbb{B}$. These elements have quantitative semantics $[\![b]\!]_q$ with the relation being $[\![b]\!](s) = \texttt{True} \iff [\![b]\!]_q(s) > 0$. Specifications ϕ are Boolean functions of the state trajectory $\zeta = (s_1, s_2, ..., s_T)$. The specification language also includes composition functions for a specification ϕ and Boolean predicate b, with the language defined as

$$\phi ::= \texttt{achieve}\ b \mid \phi\ \texttt{ensuring}\ b \mid \phi_1;\ \phi_2 \mid \phi_1\ \texttt{or}\ \phi_2$$

The description of these functions is as follows. achieve b is true when the trajectory satisfies b at least once. ϕ ensuring b is true when b is satisfied at all timesteps in the trajectory. $\phi_1;\ \phi_2$ is a sequential operator that is true when, in a given trajectory $\zeta = (s_1, s_2, ..., s_T)$, $\exists\ k > 1$ such that $\phi_1(s_1, ..., s_k)$ is true and $\phi_2(s_{k+1}, ..., s_T)$ is true. In other words, $\phi_1;\ \phi_2$ represents an ordered sequential completion of specification ϕ_1 followed by ϕ_2. Lastly, ϕ_1 or ϕ_2 is true when a trajectory satisfies either ϕ_1 or ϕ_2.

Given a specification ϕ on a Markov Decision Process $\langle S, D, A, P, T \rangle$ (MDP) defined using SPECTRL, the pipeline first compiles a task monitor $\langle Q, X, \Sigma, U, \Delta, q_0, v_0, F, \rho \rangle$ (a finite state automaton [22]) to record the completion status of tasks with monitor states Q; final monitor states F denote a satisfied trajectory. This is used to create an augmented version of the MDP $\langle \tilde{S}, \tilde{s_0}, \tilde{A}, \tilde{P}, \tilde{R}, \tilde{T} \rangle$ with an expanded state, action space and modified reward function . The task monitor provides a scoring function for trajectories in the augmented MDP to guide policy behavior.

While SPECTRL has been shown to work with trajectory-based algorithms for reinforcement learning [14], it is not immediately evident how to translate it to common RL algorithms such as DDPG [11] and PPO [18]. A simple solution would be to keep the episodic format with a trajectory $\zeta = (s_0, \cdots, s_T)$ and assign the trajectory value of SPECTRL (a function of ζ) to the final state transition in the trajectory $s_{T-1} \rightarrow s_T$ and zero for all other states. This would yield a relatively sparse reward signal that could be hard for algorithms that use the rewards r_t at time step t to quickly learn. More importantly , this maintains the trajectory ordering properties of SPECTRL in the episodic return ($\sum_{t=0}^{T} r_t$).

3 PROBLEM STATEMENT

Directly appropriating SPECTRL for our use case of imposing specifications on multi-agent problems poses significant scalability issues. Consider the cases

$$\phi_a = \texttt{achieve}(\texttt{reach}(P)); \texttt{achieve}(\texttt{reach}(Q))$$

where $[\![\texttt{reach}(P)]\!] = \texttt{True}$ when an agent reaches state P. To ease the illustration of our framework, we assume that all agents are homogeneous, *i.e.* $\mathcal{S}^i = \mathcal{S}_A, \forall i \in \mathcal{N}$. Now, the state space of the entire multi-agent system is $\mathcal{S} = (\mathcal{S}_A)^N$ for N agents (we omit m for perspicuity).

If the predicate reach was defined on the entire state $\mathcal{S}$, it would yield a specification forcing synchronization between agents. On the other hand, if reach was defined on the agent state $\mathcal{S}_A$, then it would create a *localized* specification where synchronization is not required. This would be akin to allowing individual agents to act independently of other agent behaviors.

However, using a centralized task monitor for the localized predicate would cause the number of monitor states to exponentially increase with the number of agents N and subtasks K since the possible stages of task completion would be $O(K^N)$.

To address this scalability issue, the benefits of task monitor distribution are apparent. In the case of ϕ_a above, assume reach is defined on the local state space $\mathcal{S}_A$. If each agent had a separate task monitor stored locally to keep track of the task completion stages, the new number of monitor states is now reduced to $O(NK)$.

Consider an example of robots in a warehouse. A few times a day, all robots must gather at a common point for damage inspection at the same time (akin to a global reach) to minimize the frequency of inspection (an associated cost). To ensure satisfaction of the entire specification, the reward given to an RL agent learning this objective should capture both the global and local tasks. For example, if the global reach task for the routine inspection is made local instead, the cost incurred may be larger than if it was a synchronized global objective.

Main Objective. Given a specification ϕ on a system of N agents, we wish to find policies $\Pi = \{\pi^1, \cdots, \pi^N\}$ to maximize the probability of satisfying ϕ for all agents. Formally, we seek

$$\Pi^* \in \underset{\pi^1, \cdots, \pi^N}{\arg\max} \underset{\zeta_m \sim D_\Pi}{\Pr} [\phi(\zeta_m) == \texttt{True}]$$

where D_Π is the distribution of all system rollouts when all agents collectively follow policy set Π. We emphasize that ϕ acts on the entire rollout, $\phi : Z_m \rightarrow \{0, 1\}$ and not in an agent-specific manner, $\phi' : Z^i_m \rightarrow \{0, 1\}$. This discourages agents from attempting to simply satisfy their local objectives while preventing the system from achieving necessary global ones.

4 SPECTRL IN A MULTI-AGENT WORLD

Unlike the single agent case, multi-agent problems have two major classes of objectives. Agents have individual goals to fulfill as well as collective goals that require coordination and/or global system

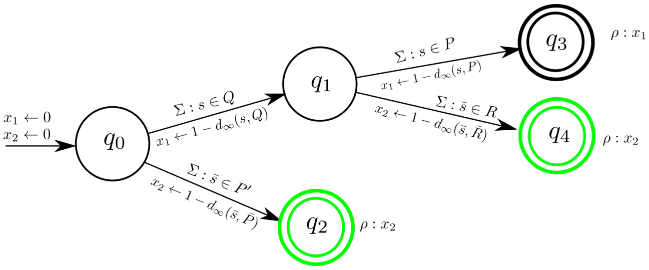
$x_1 \leftarrow 0$
$x_2 \leftarrow 0$
q_0
$\Sigma : s \in Q$
$x_1 \leftarrow 1 - d_\infty(s, Q)$
q_1
$\Sigma : s \in P$
$x_1 \leftarrow 1 - d_\infty(s, P)$
q_3
$\rho : x_1$
$\Sigma : \bar{s} \in R$
$x_2 \leftarrow 1 - d_\infty(\bar{s}, \bar{R})$
q_4
$\rho : x_2$
$\Sigma : \bar{s} \in P'$
$x_2 \leftarrow 1 - d_\infty(\bar{s}, \bar{P})$
q_2
$\rho : x_2$

Figure 1: Example Composite Task Monitor for specification ϕ_{ex} (Sec. 4) with 4 task goals denoted by Q,P,P' and R where the agent starts at q_0. Double circles represent *final* states while green circles represent *global* states. The diagram removes a state between q_1 and (q_3, q_4) as well as self-loops for ease of explanation.

knowledge. These individual goals are often only dependent on the agent-specific state s^i while collective goals require full system knowledge $\bar{s}$.

Consequently, for a multi-agent problem, we see the need for two types of predicates *viz.* local and global. Local predicates are of the form $p_{lo} : \mathcal{S}_A \rightarrow \mathbb{B}$ whereas global predicates have the form $p_{gl} : \mathcal{S} \rightarrow \mathbb{B}$ where $\mathbb{B}$ is the Boolean space. We introduce two simple extensions of `reach` [6] to demonstrate the capabilities of this distinction.

Local predicates are defined with respect to each agent and represent our individual goals. As an example, closely related to the problems observed in SPECTRL, we introduce the following local predicates for a state $s_a \in \mathcal{S}_A$,

$$[\![\texttt{reach}_{lo} x]\!](s_a) = (||s_a - x||_\infty < 1)$$

which represents reaching near location x in terms of the L_∞ norm. Now to enforce global restrictions, we introduce counterparts to these predicates that act on a global state $\bar{s} \in \mathcal{S}$.

$$[\![\texttt{reach}_{gl} \bar{x}]\!](\bar{s}) = (||\bar{s} - \bar{x}||_\infty < 1)$$

where we now have a set of locations $\bar{x} \in \mathcal{S}$.

As in SPECTRL, each of these predicates b require quantitative semantics $[\![b]\!]_q$ to facilitate our reward shaping procedure. We define these semantics as follows:

- $\texttt{reach}_{lo}$ has the same semantics as `reach` in [6] yet is defined on space $\mathcal{S}_A$.

  $$[\![\texttt{reach}_{lo} x]\!]_q(s_a) = 1 - d_\infty(s_a, x)$$

  where $d_\infty(a, b)$ represents the L_∞ distance between a and b with the usual extension to the case where b is a set.
- $\texttt{reach}_{gl}$ is defined on the state space $\mathcal{S}$ as

  $$[\![\texttt{reach}_{gl} \bar{x}]\!]_q(\bar{s}) = 1 - d_\infty(\bar{s}, \bar{x})$$

We observe that the same composition rules can apply to these predicates and we thus attempt to solve RL systems described with these compositions. As shown in Sec. 3, using a centralized SPECTRL compilation algorithm on the entire state space, even for simple sequences of tasks, leads to an explosion in monitor states. We, therefore, distribute task monitors over agents to handle scalability. Furthermore, we also need to change SPECTRL's compilation rules to handle mixed objective compositions such as[1]

$$\phi = \texttt{reach}_{lo}(P); \texttt{reach}_{gl}(Q); \texttt{reach}_{gl}(R)$$

To compile these specifications into a usable format, we utilize a *composite task monitor* as described in Sec. 5 and develop a new algorithm to achieve our goal. As an example, see Fig. 1 depicting a task monitor whose specification is:

$$\phi_{ex} = \texttt{reach}_{gl}(P') \texttt{ or } \texttt{reach}_{lo}(Q); \left[\texttt{reach}_{lo}(P) \texttt{ or } \texttt{reach}_{gl}(R)\right]$$

Here, we have 4 task goals denoted by P, Q, R and P'. The agents all start at the root node q_0. States q_2, q_3 and q_4 are all final states in the task monitor while q_2 and q_4 are global monitor states. As shown in Sec. 6, q_0 and q_1 are a *synchronization states*. While it may seem that agents only require coordination at global states, it is also necessary for the agents to have the same task transition at these synchronization states as well.

5 COMPILATION STEPS

Given a specification ϕ and the Markov game $\mathcal{M}_g$, we create a task monitor M that is distributed among agents by making agent-specific copies. This is used to create an augmented Markov game $\mathcal{M}'_g = \langle \mathcal{N}, \{\tilde{\mathcal{S}}_A\}_{i \in \mathcal{N}}, \tilde{D}, \{\tilde{A}_A\}_{i \in \mathcal{N}}, \tilde{P}, \{\tilde{R}^i\}_{i \in \mathcal{N}}, T \rangle$ on which the individual agent policies are trained.

Create Composite Task Monitor

When the types of specifications are divided into two based on the domain, the solution can be modeled with a *composite task monitor* $M_\phi = \langle Q, \tilde{X}, \tilde{\Sigma}, \tilde{U}, \tilde{\Delta}, q_0, v_0, F, \rho \rangle$. As in SPECTRL, Q is a finite set of monitor states. $\tilde{X} = X_l \cup X_g$ is a finite set of registers that are partitioned into X_l for local predicates and X_g for global predicates. These registers are used to keep track of the degree of completion of the task at the current monitor state for local and global tasks respectively.

We describe below how to use the compiled composite task monitor to create an augmented Markov game $\mathcal{M}'_g$. Each $\tilde{\mathcal{S}}_A$ in $\mathcal{M}'_g$ is an augmented state space with an augmented state being a tuple $(s_A, q, v) \in \mathcal{S}_A \times Q \times V$ where $V \in \mathbb{R}^X$ and $v \in V$ is a vector describing the register values.

$\tilde{\Delta} = \Delta_l \cup \Delta_g$ houses the transitions of our task monitor. We require that: i) different transitions are allowed only under certain conditions defined by our states and register values; and, ii) furthermore, they must also provide rules on how to update the register values during each transition. To define these conditions for transition availability, we use $\tilde{\Sigma} = \Sigma_l \cup \Sigma_g$ where Σ_l is a set of predicates over $\mathcal{S}_A \times V$ and Σ_g is a set of predicates over $\mathcal{S} \times V$. Similarly, $\tilde{U} = U_l \cup U_g$ where U_l is a set of functions $u_l : \mathcal{S}_A \times V \rightarrow V$ and U_g is a set of functions $u_g : \mathcal{S} \times V \rightarrow V$. Now, we can define $\tilde{\Delta} \subseteq Q \times \tilde{\Sigma} \times \tilde{U} \times Q$ to be a finite set of transitions that are non-deterministic. Transition $(q, \sigma, u, q') \in \tilde{\Delta}$ is an augmented transition either representing $(s^i, q, v) \xrightarrow{a^i|\Pi_{-i}} ((s^i)', q', u_l(s^i, v))$ or the form $(\bar{s}, q, v) \xrightarrow{a^i|\Pi_{-i}} (\bar{s}', q', u_g(s, v))$ depending on whether $\sigma \in \Sigma_l$ or

[1] We omit `achieve` in $\texttt{achieve}(\texttt{reach}_{lo}(P))$ and $\texttt{achieve}(\texttt{reach}_{gl}(P))$ from here on to reduce clutter; this specification is implied when we compose $\texttt{reach}(P)$ with `;` and `or`.

$\sigma \in \Sigma_g$ respectively. Let $\delta_l \in \Delta_l$ represent the former (localized) and $\delta_g \in \Delta_g$ the latter (global) transition types. Here Π_{-i} denotes the policy set of all agents except agent i. Lastly, q_0 is the initial monitor state and v_0 is the initial register value (for all agents), $F \subseteq Q$ is the set of final monitor states, and $\rho : \mathcal{S} \times F \times V \longrightarrow \mathbb{R}$ is the reward function.

Copies of these composite task monitors M are distributed over agents $\mathcal{N}$ to form the set $\{M^i\}_{i \in \mathcal{N}}$. These individually stored task monitors are used to let each agent $i \in \mathcal{N}$ keep track of its subtasks and the degree of completion of those subtasks by means of monitor state q^i and register value v^i.

Create Augmented Markov Game

From our specification ϕ we create the augmented Markov game $\mathcal{M}'_g = \langle \mathcal{N}, \{\tilde{\mathcal{S}}_A\}_{i \in \mathcal{N}}, \tilde{D}, \{\tilde{A}_A\}_{i \in \mathcal{N}}, \tilde{P}, \{\tilde{R}^i\}_{i \in \mathcal{N}}, T \rangle$ by using the compiled composite task monitor M. A set of policies $\tilde{\Pi}^*$ that maximizes rewards in $\mathcal{M}'_g$ should maximize the chance of the specification ϕ being satisfied.

Each $\tilde{\mathcal{S}}_A = \mathcal{S}_A \times Q \times V$ and $\tilde{D} = (\{s_0\}_{i \in \mathcal{N}}, q_0, v_0)$. We use Δ to augment the transitions of P with monitor transition information. Since Δ may contain non-deterministic transitions, we require the policies $\tilde{\Pi}$ to decide which transition to choose. Thus $\tilde{A}_A = A_A \times A_\phi$ where $A_\phi = \Delta$ chooses among the set of available transitions at a monitor state q. Since monitors are distributed among all agents in $\mathcal{N}$, we denote the set of current monitor states as $\bar{q} = \{q^i\}_{i \in \mathcal{N}}$ and the set of register values as $\bar{v} = \{v^i\}_{i \in \mathcal{N}}$. Now, each agent policy must output an *augmented action* $(a, \delta) \in \tilde{A}_A$ with the condition that $\delta_l = (q, \sigma_l, u_l, q')$ is possible in local augmented state $\tilde{s}_a = (s_a, q, v)$ if $\sigma_l(s_a, v)$ is $\texttt{True}$ and $\delta_g = (q, \sigma_g, u_g, q')$ is possible in global augmented state $\tilde{s} = (\bar{s}, \bar{q}, \bar{v})$ if $\sigma_g(\bar{s}, v)$ is $\texttt{True}$. We can now write the augmented transition probability $\tilde{P}$ as,

$$\tilde{P}((\bar{s}, q, v), (a, (q, \sigma, u, q')), (\bar{s}', q', u(\bar{s}, v)))) = P(\bar{s}, a, \bar{s}')$$

for transitions $\delta_g \in \Delta_g$ with $(\sigma, u) = (\sigma_g, u_g)$ and transitions $\delta_l \in \Delta_l$ with $(\sigma, u) = (\sigma_l, u_l)$. Here, we let $u_l(\bar{s}, v) = u_l(s^i, v)$ for agent i since s^i is included in $\bar{s}$. An *augmented rollout* $\tilde{\zeta}_m$ where

$$\tilde{\zeta}_m = ((\bar{s}_0, \bar{q}_0, \bar{v}_0), \bar{a}_0, ..., \bar{a}_{T-1}, (\bar{s}_T, \bar{q}_T, \bar{v}_T))$$

is formed by these augmented transitions. To translate this trajectory back into the Markov game $\mathcal{M}_g$ we can perform projection $\texttt{proj}(\tilde{\zeta}_m) = (\bar{s}_0, \bar{a}_0, ..., \bar{a}_{T-1}, \bar{s}_T,)$.

Determine Shaped Rewards

Now that we have the augmented Markov game $\mathcal{M}'_g$ and compiled our composite task monitor, we proceed to form our reward function that encourages the set of policies Π to satisfy our specification ϕ. We can perform shaping in a manner similar to SPECTRL's single-agent case on our distributed task monitor. Crucially, since reward shaping is done during the centralized training phase, we can assume we have access to the entire augmented rollout namely $\tilde{s}_t = (\bar{s}_t, \bar{q}_t, \bar{v}_t)$ at any given $t \in [0, T]$. From the monitor reward function ρ, we can determine the weighting for a complete augmented rollout as

$$\tilde{R}^i(\tilde{\zeta}_m) = \begin{cases} \rho(\bar{s}_T, q^i_T, v^i_T), & \text{if } q^i_T \in F \\ -\infty & \text{otherwise} \end{cases}$$

This reward satisfies the following property,

THEOREM 1. *For any Markov game $\mathcal{M}_g$, specification ϕ and rollout ζ_m of $\mathcal{M}_g$, ζ_m satisfies ϕ if and only if there exists an augmented rollout $\tilde{\zeta}_m$ such that i) $\tilde{R}^i(\tilde{\zeta}_m) > 0 \; \forall \; i \in \mathcal{N}$ and ii) $\texttt{proj}(\tilde{\zeta}_m) = \zeta_m$.*

The $\tilde{R}^i$ specified is $-\infty$ unless a trajectory reached a final state of the composite task monitor. To reduce the sparsity of this reward signal, we transform this into a shaped reward $\tilde{R}^i_s$ that gives partial credit to completing subtasks in the composite task monitor.

Define for a non-final monitor state $q \in Q \setminus F$, function $\alpha : \mathcal{S} \times Q \times V \longrightarrow \mathbb{R}$.

$$\alpha(\bar{s}, q, v) = \max_{(q, \sigma, u, q') \in \Delta, q \neq q'} [\![\sigma]\!]_q(\bar{s}, v)$$

This represents how close an augmented state $\tilde{s} = (\bar{s}, q, v)$ is to transition to another state $\tilde{s}'$ with a different monitor state. Intuitively, the larger α is, the higher the chance of moving deeper into the task monitor. In order to use this definition on all σ, we overload σ_l to also act on elements $\bar{s} = \{s^i\}_{i \in \mathcal{N}} \in \mathcal{S}$ by yielding for agent i, the value $\sigma_l(\bar{s}) = \sigma_l(s^i)$.

Let C_l be a lower bound on the final reward at a final monitor state, and C_u being an upper bound on the absolute value of α over non-final monitor states.

Also for $q \in Q$, let d_q be length of the longest path from q_0 to q in the graph M_ϕ (ignoring the self-loops in Δ) and $D = \max_{q \in Q} d_q$. For an augmented rollout $\tilde{\zeta}_m$ let $\tilde{s}_k = (\bar{s}_k, q^i_k, \bar{v})$ be the first augmented state in $\tilde{\zeta}_m$ such that $q^i_k = q^i_{k+1} = \cdots = q^i_T$. Then we have the shaped reward,

$$\tilde{R}^i_s(\tilde{\zeta}_m) = \begin{cases} \max\limits_{k \leq j < T} \alpha(\bar{s}_j, q^i_T, v_j) & \\ \quad + 2C_u \cdot (d_{q^i_T} - D) + C_l & \text{if } q^i_T \notin F \\ \tilde{R}^i(\tilde{\zeta}_m) & \text{otherwise} \end{cases} \tag{1}$$

THEOREM 2. *For two augmented rollouts $\tilde{\zeta}_m, \tilde{\zeta}'_m$, (i) if $\tilde{R}^i(\tilde{\zeta}_m) > \tilde{R}^i(\tilde{\zeta}'_m)$, then $\tilde{R}^i_s(\tilde{\zeta}_m) > \tilde{R}^i_s(\tilde{\zeta}'_m)$, and (ii) if $\tilde{\zeta}_m$ and $\tilde{\zeta}'_m$ end in distinct non-final monitor states q^i_T and $(q^i_T)'$ such that $d_{q^i_T} > d_{(q^i_T)'}$, then $\tilde{R}^i_s(\tilde{\zeta}_m) \geq \tilde{R}^i_s(\tilde{\zeta}'_m)$.*

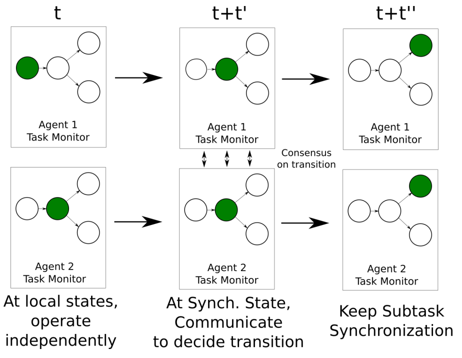

Figure 2: Overview of the DistSPECTRL process for task synchronization. Branching in the task monitor diagram denotes potential non-deterministic choices between future tasks (such as in ϕ_{ex}). Left to right represents the order of policy actions over a trajectory. Green states represent the current monitor state of that agent.

6 SUB-TASK SYNCHRONIZATION

Importance of Task Synchronization. Consider the following example specification:

$$\phi'_{1a} = \texttt{reach}_{lo}(P) \texttt{ or } \texttt{reach}_{gl}(Q)$$

where P, Q are some goals. To ensure flexibility with respect to the possible acceptable rollouts within ϕ'_{1a}, the individual agent policies π^i are learnable and the task transition chosen is dependent on the agent-specific observations. This flexibility between agents however, adds an additional possible failure method in achieving a global specification - if even a single agent attempts to fulfill the global objective while the others decide to follow their local objectives, the specification would never be satisfied.

Another example to further strengthen this notion of task synchronization is

$$\begin{aligned}\phi'_{2a} = &\left[\texttt{reach}_{lo}(P); \texttt{reach}_{gl}(Q)\ \right] \\ &\texttt{or } \left[\texttt{reach}_{lo}(P'); \texttt{reach}_{gl}(Q')\ \right]\end{aligned}$$

Here, if an agent assumed that it only needed to choose between the two local objectives $\texttt{reach}_{lo}(P)$ and $\texttt{reach}_{lo}(P')$ and not look further into the future for the presence of global objectives, a task mismatch would occur between agents who are in different branches of the task monitor *i.e.* the sub-specification $\left[\texttt{reach}_{lo}(P); \texttt{reach}_{gl}(Q)\ \right]$ vs. $\left[\texttt{reach}_{lo}(P'); \texttt{reach}_{gl}(Q')\ \right]$. We see that global objectives deeper in the sequence of specifications require prior synchronization to reaching the stage just before task completion. Thus, we see there is a marked need for task synchronization among agents as specifications become increasingly complex.

Identifying Synchronization States. As emphasized above, task synchronization is an important aspect of deploying these composite task monitors in the Markov game $\mathcal{M}_g$ with specification ϕ. We show the existence of a subset of monitor states $\textsf{Sync} \in Q$ where in order to maintain task synchronization, agents simply require a consensus on which monitor transition $\delta = (q, \sigma, u, q')$ to take. If we use Q_g to symbolize the set of global monitor states, viz. all $q \in Q$ such that $\exists(q, \sigma_g, u_g, q') \in \Delta_g$, then we see that $Q_g \subseteq \textsf{Sync}$. A valid choice for $q \in \textsf{Sync}$ with $q \notin Q_g$ is all branching states in the graph of M_ϕ with a set refinement presented in the Appendix (Sec. D).

During training, we enforce the condition that when an agent i has monitor state $q^i_t \in \textsf{Sync}$, it must wait for time $t_1 > t$ such that $q^j_{t_1} = q^i_t\ \forall j \in \mathcal{N}$ and then choose a common transition as the other agents. This is done during the centralized training phase by sharing the same transition between agents based on a majority vote.

7 MULTI-AGENT SPECIFICATION PROPERTIES

Consider a specification ϕ and let $\mathcal{N} = \{1, \ldots, N\}$ be the set of all agents with ζ_m being a trajectory sampled from the environment. $\phi(\zeta_m, n)$ is used to denote that the specification is satisfied on ζ_m for the set of agents $n \subseteq \mathcal{N}$ (i.e. $[\![\phi(\zeta_m, n)]\!] == \texttt{True}$).

MA-Distributive. Many specifications pertaining to MA problems can be satisfied independent of the number of agents. At its core, we have the condition that a specification being satisfied with respect to a union of two disjoint sets of agents implies that it can be satisfied on both sets independently. Namely if $n_1, n_2 \subset \mathcal{N}$ with $n_1 \cap n_2 = \emptyset$ then an MA-Distributive specification satisifies the following condition:

$$\phi(\zeta_m, n_1 \cup n_2) \implies \phi(\zeta_m, n_1) \wedge \phi(\zeta_m, n_2)$$

MA-Decomposable. Certain specifications satisfy a decomposibility property particular to multi-agent problems that can help in scaling with respect to the number of agents.

Say $\exists\ k \in \{1, \ldots, N-1\}$ such that

$$\phi(\zeta_m, \mathcal{N}) \implies \phi_k(\zeta_m, \mathcal{N}) = \bigwedge_{j \in \{1, \ldots, J\}} \phi(\zeta_m, n_j)$$

where

$$n_j \subset \mathcal{N}\ ,\ k \leq |n_j| < N\ ,\ J = \lfloor \frac{N}{k} \rfloor$$

$$\bigcap_j n_j = \emptyset\ ,\ \bigcup_j n_j = \mathcal{N}$$

with $\lfloor\rfloor$ representing the floor function. Each n_j is a set of at least k unique agents and $\{n_j\}_j$ forms a partition over $\mathcal{N}$.

We then call the specification ϕ *MA-Decomposable* with decomposibility factor k. Here ϕ_k can be thought of as a means to approximate the specification ϕ to smaller groups of agents within the set of agents $\mathcal{N}$. Provided we find a value of k, we can then use this as the basis of our MA-Dec scaling method to significantly improve training times for larger numbers of agents.

THEOREM 3. *All MA-Distributive specifications are also MA-Decomposable with decomposability factors $k \in \mathbb{Z}^+, 1 \leq k < N$.*

Notably all compositions of $\texttt{reach}_{gl}$ and $\texttt{reach}_{lo}$ within our language are MA-Distributive and are thus MA-Decomposable with factor $k = 2$. [2] This is far from a general property however, as one can define specifications on N robots such as $\texttt{achieve}$("collect x fruits") where each robot can carry at most x/N fruits . In this case, no single subset of agents can satisfy the specification as the total capacity of fruits would be less than x and the specification is neither MA-Distributive nor MA-Decomposable.

8 ALGORITHM

Training

Agents learn $\pi^i(s^i, v^i, q^i) = (a^i, \delta^i)$ on the augmented Markov game $\mathcal{M}'_g$ where s^i, v^i, q^i are agent-specific state, register value and task monitor state respectively. Since training is centralized, all agent task monitors receive the same global state. Based on our discussion in Sec. 6, if an agent is in any given global monitor state, we wait for other agents to enter the same state, then do the arg max task transition for all agents in the same state. In addition, at the synchronization states (Sec. 6), we perform a similar process to select the task transition. These trained *augmented policies* are then projected into policies that can act in the original $\mathcal{M}_g$.

[2] It is satisfied with $k = 1$ as well but this is the trivial case where $\texttt{reach}_{gl}$ and $\texttt{reach}_{lo}$ are equivalent.

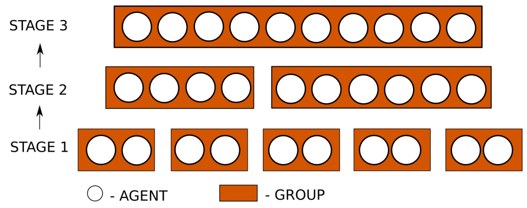

Figure 3: Example MA-Dec Scaling Process with $N = 10, k = 2, f = 2$ on an MA-Decomposable spec ϕ with decomposability factor 2. At Stage 1, $g_1 = 2$ to start with 5 groups. Next $g_2 = fg_1 = 4$ which forms 2 groups. Finally at Stage 3, $g_3 = fg_2 = 8$ which forms one group ($\mathcal{N}$).

Algorithm 1: MA-Dec Scaling

```
Given Specification $\phi$ that is MA-Decomposable with factor
  $k$ and agent set $\mathcal{N}$ of $N$ agents.
Given a function train($\phi, n_j$) that trains a policy satisfying
  $\phi$ (up to a performance metric) from a previous set of policy
  parameters for the group of agents $n_j$.
Function set_groups($g$, $\mathcal{N}$):
    // $g \in \mathbb{Z}^+$ i.e. an integer $g \geq 1$
    // Makes partition of $\mathcal{N}$ with minimum group size $g$
    if $g > |\mathcal{N}|$ then
        return $\{\mathcal{N}\}$
    else
        Initialize $j = 1, J = \lfloor \frac{N}{g} \rfloor$
        foreach $a \in \mathcal{N}$ do
            if $|n_j| < g$ or $j == J$ then
                $n_j \leftarrow n_j \cup a$
            else
                $j \leftarrow j + 1$
                $n_j = \{\}$
        end
        return $\{n_j\}_j$
Initialize Agent Policies $\Pi$, $i = 1$, $g_1 = k$
Initialize $\{n_j\}_j$ =set_groups($g_1$, $\mathcal{N}$)
while $|n_1| \leq |\mathcal{N}|$ do
    foreach $n_j \in \{n_j\}_j$ do
        Run $\Pi \leftarrow$ train($\phi, n_j$) independently of agents
          $\mathcal{N} \setminus n_j$ updating the policies of $n_j$.
    end
    $g_{i+1} = fg_i$.
    if $g_{i+1} > |\mathcal{N}|$ then
        // Already reached Final Stage with $n_1 == \mathcal{N}$
        return
    $\{n_j\}_j \leftarrow$ set_groups($g_{i+1}$, $\mathcal{N}$)
    $i \leftarrow i + 1$
end
```

Scaling MA-Decomposable Specifications

Our algorithm for scaling based off the MA-Decomposable property is shown in Alg. 1 and we name it MA-Dec scaling. At its core, we approximate the spec. ϕ by first independently considering smaller groups within the larger set of agents $\mathcal{N}$ and try to obtain a policy satisfying ϕ on these smaller groups. By progressively making the group sizes larger over stages and repeating the policy training process while continuing from the previous training stage's policy parameters , we form a curriculum that eases solving the original problem ϕ on all agents $\mathcal{N}$.

In Fig. 3 we demonstrate MA-Dec scaling for $N = 10$ agents on a spec. ϕ which is MA-Decomposable with decomposability factor 2. For this example we set the scaling parameters $k = 2$ and $f = 2$. Initially we have a min. group size $g_1 = 2$ and this is changed to $g_2 = 4$ and $g_3 = 8$ from setting the scaling factor. We increment the stage number every time all the groups of a stage have satisfied the entire specification ϕ w.r.t. their group. While separating training into stages, agents must be encouraged to move from stage i to stage $i+1$. To ensure this, we need to scale rewards based on the stage. We chose a simple linear scaling where for stage number i and time step t, each agent receives reward $r_{i,t} = ic_k + CTM_{i,t}$ where $CTM_{i,t}$ is the original composite task monitor reward at stage i and $c_k \in \mathbb{R}$ is a constant. By bounding the reward terms such that rewards across stages are monotonically increasing ($r_{i,t} < r_{i+1,t'}$) we can find a suitable c_k to be $(2D+1)C_u$ (refer Appendix Sec. B) where the terms are the same as in Eq. 1.

From setting the initial min. group size g_1 and scaling factor f, we get the total number of learning stages (T_s) as $T_s = \lfloor \log_f(N) - \log_f(k) \rfloor = O(\log_f(N))$. We build the intuition behind why MA-Dec scaling is effective in the Appendix (Sec. B), by describing it as a form of curriculum learning.

Deployment

Policies are constructed to proceed with only local information (s^i, v^i, q^i). Since we cannot share the whole system state with the agent policies during deployment yet our composite task monitor requires access to this state at all times, we allow the following relaxations: 1) Global predicates $\sigma_g(\bar{s}, v)$ enabling task monitor transitions need global state and access it during deployment. 2) Global register updates $u_g(\bar{s}, v)$ are also a function of global state and access it during deployment.

In order to maintain task synchronization, agents use on a consensus based communication method to decide task monitor transitions at global and synchronization states. If agents choose different task transitions at these monitor states, the majority vote is used as done during training.

9 EXPERIMENT SETUP

Our experiments aim to validate that the use of a distributed task monitor can achieve synchronization during the deployment of multiple agents on a range of specifications.

In addition, to emphasize the need for distribution of task monitors to alleviate the state space explosion caused by mixing local and global specifications, we include experiments with SPECTRL applied to a centralized controller.

Lastly, we provide results showing the efficacy of the MA-Dec scaling approach for larger numbers of agents when presented with a specification satisfying the MA-Decomposability property (Sec. 7).

As a baseline comparison, we also choose to run our algorithm without giving policies access to the monitor state (**no_mon**). These

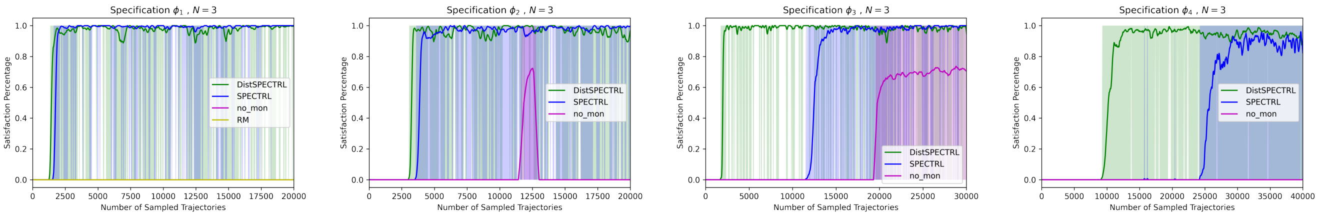
Specification ϕ_1 , $N = 3$
Specification ϕ_2 , $N = 3$
Specification ϕ_3 , $N = 3$
Specification ϕ_4 , $N = 3$

Figure 4: Satisfaction percentages on specifications ϕ_1,ϕ_2,ϕ_3 and ϕ_4 with $N = 3$ agents. The shaded regions show the maximum and minimum achieved over 5 separate evaluation runs

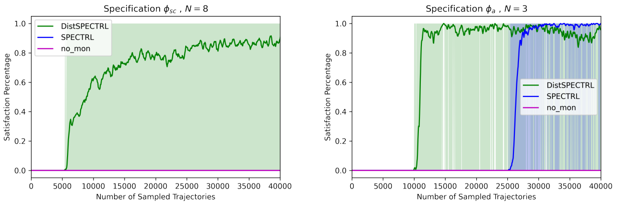
Specification ϕ_{sc} , $N = 8$
Specification ϕ_a , $N = 3$

Figure 5: Specification satisfaction percentages (left) for the StarCraft 2 specification ϕ_{sc} with $N = 8$ agents and (right) for the 3D Navigation experiments on specification ϕ_a.

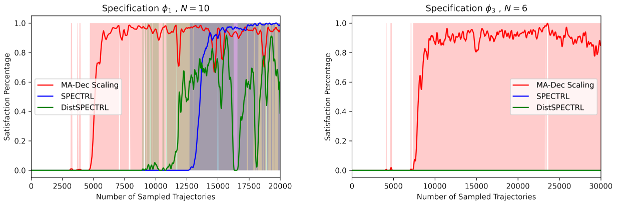
Specification ϕ_1 , $N = 10$
Specification ϕ_3 , $N = 6$

Figure 6: Specification satisfaction percentages for $N = 10$ agents on ϕ_1 (left) and $N = 6$ agents on ϕ_3 (right) comparing the MA-Dec scaling (red) to centralized SPECTRL (blue) and vanilla DistSPECTRL (green) i.e. without scaling enhancements.

are trained with the same shaped reward as DistSPECTRL and monitor transitions are set to cycle between all transitions until an available one is found. We also provide a Reward Machine baseline (**RM**) for ϕ_1 with continuous rewards since ϕ_1 is similar to the 'Rendezvous' specification in [15].

Environment. Our first set of experiments are done on a 2D Navigation problem with $N = 3$ agents. The observations ($\mathcal{S} \in \mathbb{R}^2$) used are coordinates within the space with the action space ($\mathcal{A} \in \mathbb{R}^2$) providing the velocity of the agent.

The second set of experiments towards higher dimension 3D benchmarks, represent particle motion in a 3D space. We train multiple agents ($N = 3$) in the 3D space ($\mathcal{S} \in \mathbb{R}^3$) with a 3D action space ($\mathcal{A} \in \mathbb{R}^3$) to show the scaling potential of our framework.

The final set of experiments were on a modern discrete-action MARL benchmark built in Starcraft 2 [17] with $N = 8$ agents (the "8m" map). Each agent has 14 discrete actions with a state space $\mathcal{S} \in \mathbb{R}^{80}$ representing a partial view of allies and enemies.

Algorithm Choices. For the scaling experiments (Fig. 6) we used the 2D Navigation problem with horizon $T = 500$ and the scaling parameters[3] $k = 2$ and $f = 2$. We also choose a version of PPO with a centralized Critic to train the augmented Markov Game noting that our framework is agnostic to the choice of training algorithm. The current stage is passed to the agents as an extra integer dimension. For other experiments we chose PPO with independent critics as our learning algorithm. Experiments were implemented using the RLLib toolkit [10].

Specifications

2D Navigation: The evaluated specifications are a mix of local and global objectives. The reach predicates have an error tolerance of 1 (the L_∞ distance from the goal).

$$\phi_1 = \texttt{reach}_{\texttt{gl}}(5,0); \texttt{reach}_{\texttt{gl}}(0,0)$$
$$\phi_2 = \phi_1; \texttt{reach}_{\texttt{gl}}(3,0)$$
$$\phi_3 = \texttt{reach}_{\texttt{lo}}(5,0); \texttt{reach}_{\texttt{gl}}(0,0); \texttt{reach}_{\texttt{gl}}(3,0)$$
$$\phi_4 = [\texttt{reach}_{\texttt{lo}}(3,0) \texttt{ or } \texttt{reach}_{\texttt{lo}}(5,10)]; \phi_3$$

SC2: ϕ_{sc} represents 'kiting' behaviour and is explained further in the Appendix (Sec. E).

$$\phi_{sc} = \phi_{sc_a}; \phi_{sc_a}; \phi_{sc_a}$$

where $\phi_{sc_a} = \texttt{away\_from\_enemy}_{gl}; \texttt{shooting\_range}_{lo};$

3D Environment: The following specification is considered for X-Y-Z coordinates:

$$\phi_a = \texttt{reach}_{\texttt{lo}}(5,0,0); \texttt{reach}_{\texttt{gl}}(0,0,0); \texttt{reach}_{\texttt{gl}}(3,0,0)$$

10 RESULTS

Handling Expressive Specifications. The experiments in Fig. 4 demonstrate execution when the task monitor predicates have access to the the entire system state. This provides agents with information sufficient to calculate global predicates for task monitor transitions. The overall satisfaction percentage is reported with the value 0 being an incomplete task to 1.0 being the entire specification satisfied.

While SPECTRL has often been shown to be more effective [6, 7] than many existing methods (e.g. **RM** case) for task specification, the further utility of the monitor state in enhancing coordination between agents is clearly evident in a distributed setting. The task monitor state is essential for coordination as our baseline **no_mon** is

[3] While we could start with $k = 1$, we set $k = 2$ to reduce the number of learning stages.

Table 1: Specification satisfaction percentages on convergence for Fig.4,5

Spec.	DistSpectrl	no_mon	Spectrl
ϕ_1	99.62	00.00	100.00
ϕ_2	99.05	00.00	97.38
ϕ_3	97.59	71.94	96.81
ϕ_4	97.31	00.00	90.78
ϕ_a	98.49	00.00	99.60
ϕ_{sc}	86.79	00.00	00.00

often unable to complete the entire task (even by exhaustively going through possible transitions) and global task completion requires enhanced levels of synchronization between agents.

From Table 1 we see that upon convergence of the learning algorithm, the agent is able to maintain a nearly 100% task completion rate for our tested specifications, a significant improvement in comparison to the **no_mon** case, showing the importance of the task monitor as part of a multi-agent policy.

Benefits of Distribution Over Centralization. The centralized Spectrl graphs (blue curves in Figs. 4, 5, 6) show that while distribution may not be necessary for certain specifications with few local portions (e.g. ϕ_2), concatenating them will quickly lead to learning difficulties with larger number of agents (Fig. 5, ϕ_{sc} and Fig. 6, ϕ_3). This difficulty is due in large part to state space explosion of the task monitor in these cases as is apparent by the significantly better performance of our distributed algorithm. We also remind the reader that a centralized algorithm is further disadvantageous in MARL settings due to the added synchronization cost between agents during deployment.

Scaling to Larger State Spaces. The results in Fig. 5 show promise that the DistSpectrl framework can be scaled up to larger dimension tasks as well. The 3D environment results exhibits similar behavior to the 2D case with the **no_mon** showing difficulty in progressing beyond the local tasks in the larger state space with sparser predicates. The ϕ_{sc} results also show promise in defining relevant predicates and achieving general specifications for modern MARL benchmarks.

Scaling to More Agents. Fig. 6 demonstrate the benefits of MA-Dec scaling for larger N when presented with an MA-Decomposable specification. At smaller ranges of N as well as less complex combinations of mixed and global objectives, the effect of MA-Dec scaling is less pronounced. We observe that the stage based learning is crucial for a even simple mixed specification like ϕ_3 with as little as $N = 6$ agents.

11 RELATED WORK

Multi-agent imitation learning [8, 19, 26] uses demonstrations of a task to specify desired behavior. However in many cases, directly being able to encode a specification by means of our framework is more straightforward and removes the need to have demonstrations beforehand. Given demonstrations, one may be able to infer the specification [23] and make refinements or compositions for use in our framework.

Prior work on balancing objectives between multiple agents [3] was explored for discrete action spaces and mixing two specified sets of reward functions (individual and environment). In contrast, our work is done in continuous action spaces and focuses on automatically specifying rewards for multiple local and global objectives. Other work on balancing these objectives [25] do not explicitly deal with non-Markovian tasks.

TLTL[9] is another scheme to incorporate temporal logic constraints into learning enabled controllers, although its insufficiency in handling non-Markovian specifications led us to choose Spectrl as the basis for our methodology.

Reward Machines (RMs) [2, 20, 21] are an automaton-based framework to encode different tasks into an MDP. While RMs can handle many non-Markovian reward structures, a major difference is that Spectrl starts with a logical temporal logic specification and includes with the automaton the presence of memory (in the form of registers capable of storing real-valued information). Recent work [7] shows the relative advantages Spectrl-based solutions may have over a range of continuous benchmarks.

Concurrent work has introduced the benefits of a temporal logic based approach to reward specification [5]. While experimental results are not yet displayed, the convergence guarantees of the given algorithm are promising. Since we use complex non-linear function approximators (neural networks) in our work, such guarantees are harder to provide. Reward Machines have also been explored as a means of specifying behavior in multi-agent systems [15] albeit in discrete state-action systems that lend themselves to applying tabular RL methods such as Q-learning. One may extend this framework to continuous systems by means of function approximation but to the best of our knowledge, this has not been attempted yet. Similar to our synchronization state, the authors use a defined local event set to sync tasks between multiple agents and requires being aware of shared events visible to the other agents.

While we chose PPO to train the individual agents for its simplicity, our framework is agnostic to the RL algorithm used and can be made to work with other modern multi-agent RL setups [4, 12, 13] for greater coordination capabilities.

Similar to our stage-based approach, transferring learning from smaller groups of agents to larger ones has also been explored [24].

12 CONCLUSION

We have introduced a new specification language to help detail MARL tasks and describe how it can be used to compile a desired description of a distributed execution in order to achieve specified objectives. Our framework makes task synchronization realizable among agents through the use of: 1) Global predicates providing checks for task completion that are easily computed, well-defined and tractable; 2) A monitor state to keep track of task completion; and 3) Synchronization states to prevent objectives from diverging among agents.

REFERENCES

[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel,

and Wojciech Zaremba. 2017. Hindsight Experience Replay. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/453fadbd8a1a3af50a9df4df899537b5-Paper.pdf

[2] Alberto Camacho, Rodrigo 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 (IJCAI)*. 6065–6073.

[3] Ishan Durugkar, Elad Liebman, and Peter Stone. 2020. Balancing Individual Preferences and Shared Objectives in Multiagent Reinforcement Learning. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. https://doi.org/10.24963/ijcai.2020/347

[4] JN Foerster, G Farquhar, T Afouras, N Nardelli, and SA Whiteson. 2018. Counterfactual multi-agent policy gradients. In *The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*. AAAI Press, 2974–2982.

[5] Lewis Hammond, Alessandro Abate, Julian Gutierrez, and Michael Wooldridge. 2021. *Multi-Agent Reinforcement Learning with Temporal Logic Specifications*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 583–592.

[6] Kishor Jothimurugan, Rajeev Alur, and Osbert Bastani. 2019. A Composable Specification Language for Reinforcement Learning Tasks. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d AlcheBuc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 13041–13051. http://papers.nips.cc/paper/9462-a-composable-specification-language-for-reinforcement-learning-tasks.pdf

[7] Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. 2021. Compositional Reinforcement Learning from Logical Specifications. *CoRR* abs/2106.13906 (2021). arXiv:2106.13906 https://arxiv.org/abs/2106.13906

[8] Hoang M. Le, Yisong Yue, Peter Carr, and Patrick Lucey. 2017. Coordinated Multi-Agent Imitation Learning. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*, Doina Precup and Yee Whye Teh (Eds.). PMLR, International Convention Centre, Sydney, Australia, 1995–2003. http://proceedings.mlr.press/v70/le17a.html

[9] X. Li, C. Vasile, and C. Belta. 2017. Reinforcement learning with temporal logic rewards. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 3834–3839.

[10] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. 2018. RLlib: Abstractions for Distributed Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning*. PMLR, 3053–3062. http://proceedings.mlr.press/v80/liang18b.html

[11] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2016. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1509.02971

[12] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. 2017. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6382–6393.

[13] Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. 2019. MAVEN: Multi-Agent Variational Exploration. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alche-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 7613–7624. http://papers.nips.cc/paper/8978-maven-multi-agent-variational-exploration.pdf

[14] Horia Mania, Aurelia Guy, and Benjamin Recht. 2018. Simple random search of static linear policies is competitive for reinforcement learning. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc., 1800–1809. https://proceedings.neurips.cc/paper/2018/file/7634ea65a4e6d9041cfd3f7de18e334a-Paper.pdf

[15] Cyrus Neary, Zhe Xu, Bo Wu, and Ufuk Topcu. 2021. *Reward Machines for Cooperative Multi-Agent Reinforcement Learning*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 934–942.

[16] Andrew Y. Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *In Proceedings of the Sixteenth International Conference on Machine Learning*. Morgan Kaufmann, 278–287.

[17] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob Foerster, and Shimon Whiteson. 2019. StarCraft Challenge. In *AAMAS*.

[18] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *CoRR* abs/1707.06347 (2017). arXiv:1707.06347 http://arxiv.org/abs/1707.06347

[19] Jiaming Song, Hongyu Ren, Dorsa Sadigh, and Stefano Ermon. 2018. Multi-Agent Generative Adversarial Imitation Learning. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc., 7461–7472. https://proceedings.neurips.cc/paper/2018/file/240c945bb72980130446fc2b40fbb8e0-Paper.pdf

[20] Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. 2018. Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*. 2112–2121.

[21] Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith. 2020. Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning. *arXiv preprint arXiv:2010.03950* (2020).

[22] M.Y. Vardi and P. Wolper. 1994. Reasoning about Infinite Computations. *Information and Computation* 115, 1 (1994), 1 – 37. https://doi.org/10.1006/inco.1994.1092

[23] Marcell Vazquez-Chanlatte, Susmit Jha, Ashish Tiwari, Mark K Ho, and Sanjit Seshia. 2018. Learning Task Specifications from Demonstrations. In *Advances in Neural Information Processing Systems*, Vol. 31. Curran Associates, Inc., 5367–5377. https://proceedings.neurips.cc/paper/2018/file/74934548253bcab8490ebd74afed7031-Paper.pdf

[24] Weixun Wang, Tianpei Yang, Yong Liu, Jianye Hao, Xiaotian Hao, Yujing Hu, Yingfeng Chen, Changjie Fan, and Yang Gao. [n.d.]. From Few to More: Large-Scale Dynamic Multiagent Curriculum Learning. 34, 05 ([n. d.]), 7293–7300. https://doi.org/10.1609/aaai.v34i05.6221

[25] Jiachen Yang, Alireza Nakhaei, David Isele, Kikuo Fujimura, and Hongyuan Zha. 2020. CM3: Cooperative Multi-goal Multi-stage Multi-agent Reinforcement Learning. In *International Conference on Learning Representations*. https://openreview.net/forum?id=S1lEX04tPr

[26] Lantao Yu, Jiaming Song, and Stefano Ermon. 2019. Multi-Agent Adversarial Inverse Reinforcement Learning. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, Long Beach, California, USA, 7194–7201. http://proceedings.mlr.press/v97/yu19e.html

Table 2: Specification satisfaction percentages on convergence for Fig.6, (Scaling to more Agents)

Spec. / # Agents	MA-Dec	DistSpectrl	Spectrl
ϕ_3/N =6	94.09	0.00	0.00
ϕ_1/N =6	97.83	80.67	98.96
ϕ_1/N =10	97.03	72.30	99.28

A GLOBAL TASK MONITOR CONSTRUCTION

We refer readers to the local specification compilation rules of Spectrl (defined in their Appendix). We highlight the main differences with mixed objectives here.

For a specification ϕ let Q_g the set of states with global predicates in ϕ. Note that $Q_g \subseteq \mathtt{Sync}$ as mentioned in Sec 6. Here we consider a ϕ to be global if it contains any global predicates.

$\mathtt{achieve}\ b$ when b is a global predicate

Add both states to global states Q_g.

$\phi_1;\ \phi_2$ when ϕ_2 is global

If ϕ_2 has an initial global state or $(q_0)_{\phi_2} \in (\mathtt{Sync})_{\phi_2}$ then the transition from the final state of ϕ_1 to ϕ_2 is also global. If $q_a \in F_{\phi_1}$ then $q_a \in Q_g$.

$\phi_1;\ \phi_2$ when ϕ_1 is global

It is the same as the local case with $Q_g = (Q_g)_{\phi_1}$

$\phi_1;\ \phi_2$ when both ϕ_1, ϕ_2 are global

It is the same as the local case with $Q_g = (Q_g)_{\phi_1} \cup (Q_g)_{\phi_2}$

$\phi_1 \mathtt{or}\ \phi_2$ when ϕ_2 is global

Without loss of generality, if ϕ_2 contains global states then the common start state (as part of the compilation rules of $\mathtt{or}$) is a synchronization state. $Q_g = (Q_g)_{\phi_1} \cup (Q_g)_{\phi_2}$.

B SCALING MA SPECIFICATIONS

MA-Dec Scaling can be thought of as a form of curriculum learning for MA-Distributive specifications. We progressively narrow down the valid space of parameters that satisfy the specification ϕ_k by increasing the value of k by a positive integer factor $f > 1$. Consider a set of $N = 10$ agents and an MA-Distributive specification ϕ. ϕ is also MA-Decomposable with factor 8 by Thm. 3. Since the spec. is MA-Distributive as well $\phi_8(\eta, \mathcal{N}) \implies \phi_4(\eta, \mathcal{N}) \implies \phi_2(\eta, \mathcal{N})$.

Intuitively, as shown in Fig. 7, the policy parameter Π_θ satisfying ϕ_8 will also satisfy ϕ_4 and ϕ_2 as groups of 8 agents can either be considered two groups of 4 agents or four groups of 2 agents.

Thus we position the parameter spaces as shown, and in the first stage attempt to find a parameter within the largest region satisfying $\phi_2(\mathcal{N})$. As the learning progresses, the curriculum narrows down the desired search space until we obtain the parameters satisfying the specification $\phi(\mathcal{N})$.

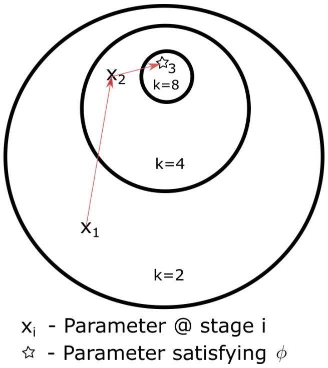

Figure 7: Scaling intuition for $N = 10, k = 2, f = 2$. We represent the policy parameter space and the respective placement of parameters that satisfy ϕ_k for various values of k. The arrows show the direction we proceed searching for parameters in our scaling process.

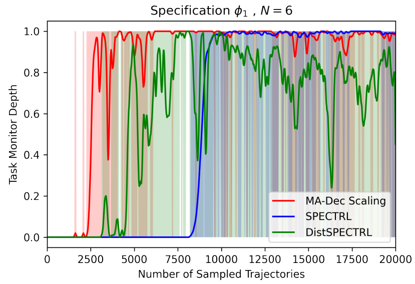

Figure 8: Specification satisfaction percentages for $N = 6$ agents on ϕ_1 comparing the MA-Dec scaling method (red) to centralized Spectrl (blue) and vanilla DistSpectrl (green).

Calculating Scaling constant c_k

We want all rewards at stage i to be less than the rewards at stage $i + 1$ to prevent local optima from arising where an agent is not incentivized to progress to the next stage. Assuming that the final reward at all stages is also upper bounded by C_u (as is α).

$$\begin{aligned} r_{i,t} &\le r_{i+1,t'} \qquad \forall t, t' \\ \implies ic_k + CTM_{i,t} &\le (i+1)c_k + CTM_{i+1,t'} \\ \implies \max(ic_k + CTM_{i,t}) &\le \min((i+1)c_k + CTM_{i+1,t'}) \end{aligned}$$

Since $c_k \in \mathbb{R}$ is a constant we get

$$\begin{aligned} \max(CTM_{i,t}) - \min(CTM_{i+1,t'}) &\le c_k \\ \implies C_u - (-2DC_u) &\le c_k \end{aligned}$$

Thus a suitable c_k is $(2D + 1)C_u$.

C IMPLEMENTATION

Computational resources

All experiments were run on a Intel Xeon Gold 2.10 Ghz 64-core machine with 252 GB of RAM. Individual experiments used no more than 16 cores at a time with experiments involving a hyperparameter search taking 2 cores each.

Hyperparameters

A single 2 layer neural network with 256 nodes each and a tanh activation function was used. The learning rate was varied from 1×10^{-3} to 1×10^{-5} over 2×10^{7} iterations.

We used a grid search on hyperparameters for all experiments.

Table 3: Hyperparameters used for grid search

Hyperparameter	Ranges
Batch Size	$[10000, 20000]$
Initial Learning Rate	$[10^{-3}, 10^{-4}, 10^{-5}]$
Entropy Coefficient	$[0, 0.00176]$

Metrics

The specification satisfaction is reported with value from 0 being no sub-task completed to 1.0 being the entire specification satisfied. For more details on the compilation rules we refer readers to Sec. A.

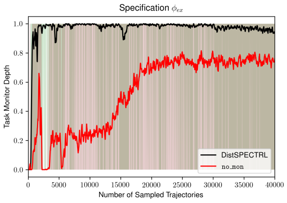

Figure 9: Specification satisfaction percentages for the task monitor shown in Fig. 1 with ϕ_{ex} = $\texttt{reach}_{gl}(10,10); \texttt{or}\ [\texttt{reach}_{lo}(3,0); [\texttt{reach}_{lo}(10,10)\ \texttt{or}\ \texttt{reach}_{gl}(5,0)]]$

D SUBTASK SYNCHRONIZATION

The identification of local synchronization states $q \in \texttt{Sync}$, where $q \notin Q_g$ is as follows:

(1) Select all branching states in the graph of M_ϕ.
(2) Remove those with all branches local and disconnected. That is, all monitor states in these branches that only have local transitions δ_l.
(3) Remove all those whose branches rejoin at some state (the rejoin point) and have all paths from branching state to the rejoin point not include any global monitor states. That is, if we consider only the subgraph of M_ϕ starting from the branching state, the rejoin point should have no ancestors which are global monitor states.

E ENVIRONMENTS

2D Environment

The environment follows first order dynamics in a 2D space ($\mathcal{S} \in \mathbb{R}^2$). The action space ($\mathcal{A} \in \mathbb{R}^2$) provides the velocity of the agent in the space. Agents are initialized in a line below their reference goals at a Y-coordinate uniformly sampled between $(2, 3)$.

3D Environment

The environment follows first order dynamics in a 3D space ($\mathcal{S} \in \mathbb{R}^3$). The action space ($\mathcal{A} \in \mathbb{R}^3$) provides the velocity of the agent in the space. Agents are initialized below the goal in the X-Y plane and with a random Y,Z coordinate uniformly sampled between $(2, 3)$.

StarCraft 2

Starcraft 2 [17] experiments used the "8m" map with 8 controllable marines and 8 enemy AI-controlled marines. Each agent had state space $\mathcal{S} \in \mathbb{R}^{80}$ and 14 discrete actions. `away_from_enemy` defines a predicate that is true when the agent cannot be shot by the enemy. `shooting_range` defines a predicate that is true when the agent can shoot the enemy (but can also be shot as well). $\texttt{away\_from\_enemy}_{gl}$ being true implies that all the agents are together away from the enemy at once (in a synchronized manner).

To define these predicates we make use of two indicators provided by the Starcraft 2 environment. The first being $\texttt{shooting\_range\_ind} \in \{0, 1\}$ which is 1 when the enemy within shooting range and 0 otherwise. The other is $\texttt{dist\_to\_enemy} \in [0, 1]$ being the normalized distance to an enemy which is 0 when the enemy is not visible (the observation radius is larger than the shooting range) .

The quantitative semantics of these new predicates are then

$$\begin{aligned} [\![\texttt{away\_from\_enemy}]\!]_q =& (1 - \texttt{shooting\_range\_ind}) * (\epsilon) + \\ & \texttt{shooting\_range\_ind} * (\texttt{dist\_to\_enemy} - \epsilon) \\ [\![\texttt{shooting\_range}]\!]_q =& (1 - \texttt{shooting\_range\_ind}) * (-\epsilon) + \\ & \texttt{shooting\_range\_ind} * (\texttt{dist\_to\_enemy} - \epsilon/10) \end{aligned}$$

where $\epsilon \in [0, 1]$ is a real-value representing the error tolerance set to 0.5 in the Starcraft experiments.

To augment our discrete action space Markov Game for the centralized SPECTRL comparison, we included an additional agent with access to the full system state. This centralized controller was used to choose between the available task monitor transitions.

F PROOFS

Proof of Theorem 1

The proof follows the exact outline as in SPECTRL since the language composition and compilation rules are equivalent in the necessary steps. We repeat their arguments here for clarity. First, the following lemma follows by structural induction:

LEMMA 4. *For* $\sigma \in \Sigma$, $[\![\sigma]\!](s, v) = \texttt{True} \iff [\![\sigma]\!]_q(s, v) > 0$.

Next, let G_M denote the underlying state transition graph of a task monitor M. Then,

LEMMA 5. *The task monitors constructed by our algorithm satisfy the following properties:*

(1) *The only cycles in G_M are self loops.*

(2) *The finals states are precisely those states from which there are no outgoing edges except for self loops in G_M.*
(3) *In G_M, every state is reachable from the initial state and for every state there is a final state that is reachable from it.*
(4) *For any pair of states q and q', there is at most one transition from q to q'.*
(5) *There is a self loop on every state q given by a transition $(q, \top, u, q)$ for some update function u where $\top$ denotes the true predicate.*

The first three properties ensure progress when switching from one monitor state to another. The last two properties enable simpler composition of task monitors. The proof follows by structural induction. Theorem 1 now follows by structural induction on ϕ and Lemmas 4 and 5.

Proof of Theorem 2

i) Let $\tilde{\zeta}_m, \tilde{\zeta}'_m$ be two augmented rollouts such that $\tilde{R}^i(\tilde{\zeta}_m) > \tilde{R}^i(\tilde{\zeta}'_m)$.

- Case A. Both $\tilde{\zeta}_m, \tilde{\zeta}'_m$ end in final monitor states. Here $\tilde{R}^i_s(\tilde{\zeta}_m) = \tilde{R}^i(\tilde{\zeta}_m) > \tilde{R}^i(\tilde{\zeta}'_m) = \tilde{R}^i_s(\tilde{\zeta}'_m)$.
- Case B. $\tilde{\zeta}_m$ ends in a final monitor state but $\tilde{\zeta}'_m$ does not. Here

$$\begin{aligned}
\tilde{R}^i_s(\tilde{\zeta}'_m) = & \max_{k \le j < T} \alpha(\bar{s}_j, q^i_T, v_j) \\
& + 2C_u(d_{q^i_T} - D) + C_l \\
\le & \max_{k \le j < T} \alpha(\bar{s}_j, q^i_T, v_j) - 2C_u + C_l && (d_{q^i_T} - D \le -1) \\
\le & C_l && (C_u \ge \alpha, C_u \ge 0) \\
\le & \tilde{R}^i(\tilde{\zeta}_m) && (C_l \le \tilde{R}^i \; \forall \; i \in \mathcal{N}) \\
= & \tilde{R}^i_s(\tilde{\zeta}_m) && (q^i_T \in F)
\end{aligned}$$

- Case C. $\tilde{\zeta}_m$ ends in a non-final monitor state. Here $\tilde{R}^i(\tilde{\zeta}_m) = -\infty$ and $\tilde{R}^i(\tilde{\zeta}'_m) = -\infty$ as well.

(ii) if $\tilde{\zeta}_m$ and $\tilde{\zeta}'_m$ end in distinct non-final monitor states q^i_T and $(q^i_T)'$ such that $d_{q^i_T} > d_{(q^i_T)'}$, then $\tilde{R}^i_s(\tilde{\zeta}_m) \ge \tilde{R}^i_s(\tilde{\zeta}'_m)$.

Here the trajectories vary in only one agent's monitor state.

$$\begin{aligned}
\tilde{R}^i_s(\tilde{\zeta}_m) = & \max_{k \le j < T} \alpha(\bar{s}_j, q^i_T, v_j) + C_l \\
& + 2C_u(d_{q^i_T} - D) \\
\ge & \max_{k \le j < T} \alpha(\bar{s}_j, q^i_T, v_j) + C_l && (d_{q^i_T} \ge d'_{q^i_T} + 1) \\
& + 2C_u(d'_{q^i_T} - D) + 2C_u \\
\ge & C_u + C_l && (C_u > |\alpha| \implies C_u > -\alpha) \\
& + 2C_u(d'_{q^i_T} - D) \\
\ge & \max_{k \le j < T} \alpha(\bar{s}'_j, (q^i_T)', v'_j) + C_l && (C_u > \alpha) \\
& + 2C_u(d'_{q^i_T} - D) \\
= & \tilde{R}^i_s(\tilde{\zeta}'_m)
\end{aligned}$$

Proof of Theorem 3

Given ϕ being MA-Distributive, then for two disjoint sets of agents $n_1, n_2 \subset \mathcal{N}$

$$\phi(\zeta_m, n_1 \cup n_2) \implies \phi(\zeta_m, n_1) \wedge \phi(\zeta_m, n_2)$$

Given a value $k \in \mathbb{Z}^+, 1 \le k < N$ we can create a group of agent sets $\{n_j\}_{j \in 1,\ldots,J}$ forming a partition of $\mathcal{N}$ with minimum group size k using the $\texttt{set\_groups}(k, \mathcal{N})$ function in Alg.1. Now

$$\begin{aligned}
\phi(\zeta_m, \bigcup_{j \in 1,\ldots,J} n_j) & \implies \phi(\zeta_m, n_1) \wedge \phi(\zeta_m, \bigcup_{j \in 2,\ldots,J} n_j) \\
& \implies \phi(\zeta_m, n_1) \wedge \phi(\zeta_m, n_2) \wedge \phi(\zeta_m, \bigcup_{j \in 3,\ldots,J} n_j) \\
& \implies \bigwedge_{j \in \{1,\ldots,J\}} \phi(\zeta_m, n_j)
\end{aligned}$$

Thus ϕ is also MA-Decomposable with factors $k \in \mathbb{Z}^+, 1 \le k < N$.