24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

Extended Markov Games to Learn Multiple Tasks in Multi-Agent Reinforcement Learning

Borja G. Leon and **Francesco Belardinelli** [1]

Abstract. The combination of Formal Methods with Reinforcement Learning (RL) has recently attracted interest as a way for single-agent RL to learn multiple-task specifications. In this paper we extend this convergence to multi-agent settings and formally define Extended Markov Games as a general mathematical model that allows multiple RL agents to concurrently learn various non-Markovian specifications. To introduce this new model we provide formal definitions and proofs as well as empirical tests of RL algorithms running on this framework. Specifically, we use our model to train two different logic-based multi-agent RL algorithms to solve diverse settings of non-Markovian *co-safe* LTL specifications.

1 Introduction

Reinforcement Learning is increasingly becoming a dominant paradigm for training autonomous agents in unknown and high-dimensional scenarios [23, 29]. Its combination with temporal logic from Formal Methods [8, 35], holds considerable promise to help learning agents tackle several tasks as a single specification, including safety constrains, which play a major role in many real-world scenarios, such as autonomous driving [30], autonomous robots [15] or network packet delivery [37]. Additionally, temporal logic offers a simple but efficient way to use Reinforcement Learning (RL) algorithms to find optimal policies in non-markovian settings [3, 5, 8].

These real-world scenarios, in which agents have to work in an environment where there are other learning agents, are naturally modelled as a multi-agent system (MAS). Unfortunately, MAS have several challenges on their own, such as scalability with the number of agents as well as issues pertaining to non-stationarity, i.e, that in MAS probabilities attached to transitions do not depend solely on the single agent's actions and the corresponding transition probabilities.

This dependency on other agents' policies means that the notion of optimality, central to Markov Decision Processs (MDPs), needs to be encapsulated in a sort of equilibrium to solve the optimization problem. In this context, Nash equilibrium [13], where each agent's policy is the best response to the others', is one of the most common solution concepts. Multi-Agent RL (MARL) has the potential to overcome scalability shortcomings by showing promising results in applications such as robot swarms [14] and the aforementioned autonomous vehicles [6].

Despite the growing success of MARL, also due to the adoption of neural networks in Deep (RL) [31], there is still little work on combining these multi-agent learning techniques with formal methods. This is the long-term challenge that we here start to address.

[1] Imperial College London, UK, email: bg19@imperial.ac.uk, francesco.belardinelli@imperial.ac.uk

Contribution. This paper focus on formally extending Markov Games (MGs), the mathematical model that is traditionally used in MARL, to build a new general model, i.e, not focused solely in one kind of multi-agent game, that allows multiple learning agents to concurrently fulfill various non-Markovian specifications in multi-agent settings. To support our model with empirical evidence, we also extended two logic-based RL algorithms to multi-agents systems in order to show how various learning agents can fulfill different types of non-Markovian specifications expressed in *co-safe-* Linear-time Temporal Logic (LTL). Our results are promising and point to interesting future work we discuss in Sec. 6.

Related Work. There is a growing interest towards the interactions of Formal Methods and Reinforcement Learning. In [1] model checking of temporal logics is adopted to build a shielding system that is meant to prevent a learning agent from taking dangerous actions. In [22] verification techniques are applied to abstract MDPs to check abstracted policies learned with RL. Related works (see, e.g., [38, 36]) pursuit similar verification goals in the sub-field of Inverse Reinforcement Learning, where reward functions are initially unknown and Formal Methods enable some degree of safety on the extracted rewards. Closer to the present contribution, [24] tackles MARL with rewards expressed in temporal logic. Specifically, they enforce Signal Temporal Logic (STL) specifications in a mini-max game extension of the popular decentralized Independent Deep Q-Learning (I-DQN) algorithm for adversarial settings. Our approach differs under two key aspects. First, we provide a formal definition on how to use Formal Methods (FM) in order train MARL policies to solve Non-Markovian Reward Game (NMRG), which is not restricted to a specific temporal logic, such as STL. Second, our approach is meant to cover a wide range of multi-agent scenarios, not only adversarial as in [24].

Solving multiple task-specifications expressed in temporal logic has also been previously addressed in [8], where authors show how an RL agent is able to learn non-Markovian rewards expressed in LTL over finite traces (LTL_f) and Linear Dynamic Logic over finite traces (LDL_f), by using reward shaping [25]. Toro et al. [35] also tackle the problem of learning multiple tasks expressed in *co-safe* LTL. They introduce an extension of Q-learning called LTL Progressionfor Off-Policy Learning (LPOPL) that makes use of the additional information provided by the temporal logic specifications in order to achieve an agent's goal. Our work focus on extending what [8, 35] developed for the single-agent framework to multi-agent systems, where reinforcement learning guided by temporal specifications could prove to be an effective tool to address some of the complex challenges mentioned above, including scalability, stationarity or equilibrium of solutions.

A document with full proofs is available at [19]

2 Background

In this section we recall preliminary notions on Markov games, the linear-time temporal logic LTL, and Q-learning.

Markov Games. A single-agent RL task is typically modelled as a MDP, which means that, in order to work with the performing algorithms from RL, the reward given to the learning agent need to be Markovian, i.e., it must depend only on the last state and action taken.

A multi-agent RL task, i.e, a problem-solving cooperative, competitive or adversarial setting where there are two or more learning agents whose goal is to individually collect the maximum possible reward by interacting with the environment, is modeled as a Markov game [20].

Definition 1 (MG)**.** *A* Markov game *is a tuple* $\mathcal{G} = \langle N, S, A, P, R, \gamma\rangle$ *where:*

- *N is the set of n agents.*
- *S is a finite set of states.*
- *A is the action set where each A_i is the action set of agent $i \in N$. A^t refers to the joint action taken by all agents in state s_t.*
- *$P : S \times A_1 \times \cdots \times A_n \times S \to [0, 1]$ is the transition probability function that returns the probability of transitioning to a new state given the previous state and the actions taken by all agents.*
- *$R_i : S \times A_1 \times \ldots \times A_N \to \mathbb{R}$ is the reward function of agent i and $R = \{R_1, \ldots, R_N\}$ is the set of reward functions.*
- *$\gamma \in (0, 1]$ is the discount factor that is used to discount future rewards.*

Markov games can be thought of as multi-agent extension of MDPs where actions are chosen and executed simultaneously, thus new states depend on the action taken by each single agent. Notice also that we are working in a fully observable setting, which means that individual observation functions are trivial, and therefore omitted for sake of simplicity.

Linear Time Temporal Logic [26] is an expressive and well-studied logic-based language to specify properties of MAS modelled as Markov games. Formulae in LTL are built from a set $\mathcal{AP}$ of atoms, by using Boolean and temporal operators as follows:

$$\varphi \ ::= \ p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \mathrm{U} \varphi_2 \tag{1}$$

where $p \in \mathcal{AP}$, $\bigcirc$ is the *next* operator, and U is the *until* operator. We use the standard abbreviations: *eventually* $\diamond\varphi \equiv \top \mathrm{U}\varphi$; *always* $\Box\varphi \equiv \neg \diamond \neg\varphi$; *weak next* $\bullet\varphi \equiv \neg \bigcirc \neg\varphi$ (when interpreted over finite traces $\neg \bigcirc \varphi \not\equiv \bigcirc\neg\varphi$); and $Last \equiv \bullet false$ denotes the end of the trace (if finite).

The semantics of LTL formulae over $\mathcal{AP}$ are defined over (finite or infinite) sequences $\sigma = \langle\sigma_0, \sigma_1, \sigma_2, \ldots\rangle \in (2^{\mathcal{AP}})^\omega \cup (2^{\mathcal{AP}})^+$ of truth assignments for the atoms in $\mathcal{AP}$, where each σ_i is a truth assignment to each atom in $\mathcal{AP}$. By $p \in \sigma_i$, for atom $p \in \mathcal{AP}$, we mean that p is true in σ_i. The length of a sequence σ is denoted as $|\sigma|$, with $|\sigma| = \omega$ if σ is infinite. For $i \leq |\sigma|$, let $\sigma_{\geq i}$ be the suffix $\sigma_i, \sigma_{i+1}, \ldots$ of σ starting at σ_i and $\sigma_{\leq i}$ its prefix $\sigma_0, \ldots, \sigma_i$.

We can now define formally when a sequence σ *satisfies* an LTL formula φ at time $i \geq 0$, denoted by $\langle\sigma, i\rangle \models \varphi$, as follows (clauses for Boolean connective are immediate and thus omitted):

$$\begin{array}{ll}
\langle\sigma, i\rangle \models p & \text{iff } p \in \sigma_i \\
\langle\sigma, i\rangle \models \bigcirc\varphi & \text{iff } |\sigma| \geq i+1 \text{ and } \langle\sigma, i+1\rangle \models \varphi \\
\langle\sigma, i\rangle \models \varphi_1 \mathrm{U}\varphi_2 & \text{iff for some } j, i \leq j \leq |\sigma| \text{ and } \langle\sigma, j\rangle \models \varphi_2 \text{, and} \\
& \text{for all } k, i \leq k < j \text{ implies } \langle\sigma, k\rangle \models \varphi_1
\end{array}$$

A sequence σ is said to satisfy or model φ iff $\langle\sigma, 0\rangle \models \varphi$. Notice that, whenever σ is infinite, we obtain the standard semantics for LTL.

We mentioned in Sec. 1 two variants of LTL that have been used in the latest works extending single-agent RL with temporal logic [8, 35]. Both are motivated by the fact that, in RL, agents are typically trained over finite episodes, but in different way. The former is LTL *over finite traces* (LTL_f), which is standard LTL interpreted over finite traces only, instead of infinite ones [9]. The latter is *co-safe* LTL [17], which – like LTL – is interpreted over infinite traces, but its syntax is restricted as follows:

$$\varphi \ ::= \ p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \mathrm{U}\varphi_2 \tag{2}$$

The restricted syntax of *co-safe* LTL is meant to guarantee that formulas, if true, are true after a finite number of steps. We can show that LTL_f and *co-safe* LTL are actually equivalent, with the former restricting the semantics while the later restricts the syntactic. The equivalence is stated in Corollary 1 below.

First, by next result every *co-safe* LTL formula is true iff it is satisfied by a finite sequence.

Proposition 1. *Let $\sigma \in (2^{\mathcal{AP}})^\omega$ be an infinite sequence, and φ a* co-safe *LTL formula. Then,*

$$\sigma \models \varphi \quad \textit{iff} \quad \textit{for some } i \geq 0, \sigma_{\leq i} \models \varphi \tag{3}$$

Next we prove that formulae in LTL_f can be translated into *co-safe* LTL in a truth-preserving manner. First of all, we assume operators $\Box$ and $\bullet$ as primitive and consider negation on atoms only. We also assume atom $Last$ as primitive, only true in the last element in σ. Then, consider translation τ from LTL into *co-safe* LTL, which commutes with all operators, and such that

$$\begin{array}{rcl}
\tau(\Box\varphi) & = & \tau(\varphi)\mathrm{U}(Last \wedge \tau(\varphi)) \\
\tau(\bullet\varphi) & = & \neg Last \to \bigcirc\tau(\varphi)
\end{array}$$

Proposition 2. *Let $\sigma \in (2^{\mathcal{AP}})^+$ be a finite sequence, φ an LTL formula,*

$$\sigma \models \varphi \quad \textit{iff} \quad \textit{for every } \sigma' \in (2^{\mathcal{AP}})^\omega, \sigma \cdot \sigma' \models \tau(\varphi) \tag{4}$$

where $\cdot$ is string concatenation.

By combining Proposition 1 and 2, we obtain the following result.

Corollary 1. *There are truth-preserving (polynomial) translations between LTL_f and* co-safe *LTL.*

In this paper we work with specifications expressed in *co-safe* LTL, algorithm LPOPL introduced in [35], which is based on *co-safe* LTL. Nevertheless, as can be inferred from Sec. 3.2, this approach is not restricted to LTL and its variants, but applies to any formal language whose specifications can be transformed into a Deterministic Finite Automaton (DFA), that is, a finite-state machine that accepts or rejects finite strings of symbols [27].

Definition 2 (DFA). *A* deterministic finite automaton *is a tuple* $\mathcal{A} = \langle\Sigma, Q, q, \delta, F\rangle$*, where* Σ *is the* input alphabet, Q *is the set of* states *with* initial state $q_0 \in Q$, $\delta : \Sigma \times Q \to Q$ *is the transition function, and* $F \subseteq Q$ *is the set of* final states.

For the mathematical framework presented in Sec. 3.2 to work, we will need to transform *co-safe* LTL specifications to a correspondent DFA. From [18] we know that for any co-safe formula φ_k we can build a correspondent DFA $\mathcal{A}_\varphi = \langle 2^{\mathcal{AP}}, Q_\varphi, q_{\varphi 0}, \delta_\varphi, F_\varphi \rangle$ that accepts exactly the finite traces that satisfy φ.

From here on, *co-safe* LTL specifications will be referred just as LTL specifications without abuse of notation since we proved that both LTL_f and *co-safe* LTL are actually equivalent.

LTL progression. LTL formulae can be progressed [4] along a sequence of truth assignments. In MARL, that means the formulae representing the specifications to be learned by the agents can be updated during an episode to reflect those requirements from the formulae that have been satisfied by the current history of states. Thus, progressed formulae include only those parts of the original formulae that remain to be satisfied. For example, given the formula $\diamond(p \wedge \diamond q)$ (eventually p and then eventually q) can be progressed to $\diamond q$ once the agents reaches a state where p is true. We now introduce a formal definition of progression similarly to [4].

Definition 3. *Given an* LTL *formula* φ *and a truth assignment* σ_k *over* $\mathcal{AP}$*,* $prog(\sigma_k, \varphi)$ *is defined as follows:*

- $\operatorname{prog}(\sigma_k, p) = true$ *if* $p \in \sigma_k$*, where* $p \in \mathcal{AP}$
- $\operatorname{prog}(\sigma_k, p) = false$ *if* $p \notin \sigma_k$*, where* $p \in \mathcal{AP}$
- $\operatorname{prog}(\sigma_k, \neg\varphi) = \neg \operatorname{prog}(\sigma_k, \varphi)$
- $\operatorname{prog}(\sigma_k, \varphi_1 \wedge \varphi_2) = \operatorname{prog}(\sigma_k, \varphi_1) \wedge \operatorname{prog}(\sigma_k, \varphi_2)$
- $\operatorname{prog}(\sigma_k, \bigcirc\varphi) = \varphi$
- $\operatorname{prog}(\sigma_k, \varphi_1 \cup \varphi_2) = \operatorname{prog}(\sigma_i, \varphi_2) \vee (\operatorname{prog}(\sigma_k, \varphi_1) \wedge \varphi_1 \cup \varphi_2)$

Example 1. *As running example throughout the paper, as well as for the experimental evaluation in Sec. 5, we consider a Minecraft-like grid world similar to the one introduced in [2] but extended with multiple learning agents. In this multi-agent scenario, the learning agents can interact with objects, extract raw materials from their environment and use them to manufacture new objects. This scenario also includes a set of features and events that are detectable by the agents: {got_wood, used_toolshed, used_workbench, got_grass, used_factory, got_iron, used_bridge, used_axe, at_shelter} with obvious interpretations. This set becomes the set of atoms* $\mathcal{AP}$ *in our example. By using these atoms we can specify long-term goals in* co-safe LTL*, e.g., the specification of making shears can be expressed as:*

$$\begin{aligned}\varphi_{shears} \triangleq\ & \diamond \big(got\_wood \wedge \diamond used\_workbech\big) \wedge \\ & \diamond \big(got\_iron \wedge \diamond used\_workbech\big)\end{aligned}$$

Notice that wood and iron can be collected either way and that one workbench usage is enough to fulfill the task as long as it is done after collecting both items. We call this kind of specification an interleaving specification. We suppose that agents have to collaboratively fulfill a number of multi-task specifications, such as make shears by collecting wood and iron and using the workbench. We can model such a scenario as a Markov game, by defining S *as a grid map representing the positions of the agents in* N *and the objects within the grid. Then, the actions in* A_i *available to each agent* $i \in N$ *will be the set* $\{Up, Down, Left, Right, Wait\}$*. Further,* P *will represent the probability of reaching a new state, i.e, a new distribution on the grid map, given the previous distribution and the joint action taken by the agents. The reward* R_i *will be the same for all agents and a positive reward would be given by the system when the relevant specification is satisfied. Note that, in order for this model to be Markovian, the satisfaction of the specifications must depend only on the last state and action taken. Finally, the discount factor* γ *will have a value between 0 or 1, depending on how important we want future rewards to be for the agents.*

Q-learning [7] is an off-policy model-free RL algorithm that is at the core of all the multi-agent methods we use in our experiments in Sec. 5. By off-policy we mean that it is a learning method that aims for a target policy while using a behavior policy, and by model-free that the algorithm does not build a model of the environment to find the optimal policy, where a *policy* is a mapping from states to actions.

Q-learning works by initializing the Q-values of every state-action pair to any value (usually zero). Then, at every time-step, the algorithm uses a behaviour policy to execute an action a in the current state s, which leads to a new state s' and reward r returned by the environment. Given a learning rate α and a future-reward discount factor γ, the estimated Q-value $Q(s,a)$ is then updated as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q\left(s', a'\right) - Q(s,a) \right] \quad (5)$$

Intuitively, Equation 5 means that Q-values, i.e., the "quality" of a given state-action pair, are updated according to the weighted sum of their current values and the difference between the new observed reward outcome plus its expected value and the expected value given the original state and action taken.

This algorithm is guaranteed to converge to the optimal Q-values as long as every state-action pair is visited infinitely often [7]. A method to fulfill this requirement is to set the behavioral policy to be ϵ-greedy on the target policy. That is, for each time-step, the behavior policy selects a random action with probability ϵ and the action with the highest Q-value (the one given by the target policy) with probability $1 - \epsilon$.

Deep Q-learning. A popular technique to address the Q-learning algorithm is by using a table that stores every single state-action pair. This, however, is impractical for environments with large (or infinite) state spaces. Deep Q-learning instead makes use of experience replay and deep neural networks parameterised by θ, where θ are the weights of the neural network to represent an action-value function for a given state, that is, by using neural networks as function approximator of the table from the original algorithm. Deep Q-Networks (DQNs) were introduced in [23], where the algorithm makes use of a replay memory to store transition tuples of the form $\langle s, a, r, s' \rangle$. Function θ is learnt by sampling batches of transitions from the replay memory and minimizing the following loss function:

$$\mathcal{L}(\theta) = \sum_{i=1}^{M} \left[\left(y_i^{\mathrm{DQN}} - Q(s,a;\theta) \right)^2 \right] \quad (6)$$

where $y^{\mathrm{DQN}} = r + \gamma \max_{u'} Q\left(s', a'; \theta^-\right)$ and θ^- are the parameters of a target network that are periodically updated (copied) from the parameters θ of the training network and kept fixed for a number of iterations. Note that even when Deep Q-learning cannot guarantee to find the optimal policy, it is highly effective on large state spaces [23].

LPOPL. LPOPL [35] is a reward-tailored Q-learning function designed for single-agent reinforcement learning problems with *co-safe* LTL specifications. We will employ a decentralized extension from the deep learning version of this algorithm in our experiments.

Independent Q-learning. Independent Q-learning [33] is perhaps the most spread approach in MARL. It basically decomposes a multi-agent problem into a collection of simultaneous single-agent problems that share the same environment. Even when this approach does not address the non-stationarity problem introduced by the changing policies of the other agents, it nonetheless commonly serves as a strong benchmark for a range of MAS [32, 34, 28].

3 Non-Marvokian Multi-Agent Specifications

Reinforcement Learning is designed to work under Markovian reward models. The main limitation of this kind of modeling is the Markovian assumption, i.e., rewards depend only on the last state visited and action taken. Thus we cannot adopt reward functions that capture conditional temporally extended properties on vanilla Markov games, such as LTL specifications. In this section we detail how to handle non-Markovian specifications with RL in general-purpose game systems.

3.1 Non-Markovian Reward Games

In this section we introduce the concept of non-Markovian reward game, which is similar to Markov games, but where the reward depends on the history of states visited and actions taken.

Definition 4 (NMRG)**.** *A non-Markovian reward game is a tuple* $\mathcal{G} = \langle S, N, A, P, \overline{R}, \gamma\rangle$ *where:*

- S, N, A, P *and* γ *are defined as for a Markov game in Def. 1;*
- *The reward* $\overline{R}$ *is a set of real-valued functions over state-action histories in* $(S \times A_1 \times \cdots \times A_N)^*$ *(which we will refer as traces), i.e., for every* $i \in N$, $\overline{R}_i : (S \times A_1 \times \cdots \times A_N)^* \to \mathbb{R}$.

By Def. 4, NMRGs allow us to define rewards that depend on a full trace rather than just the last state and action taken. We are also working with LTL specifications so we embed them into the NMRG.

Definition 5. *A NMRG with* co-safe LTL *specifications is a tuple* $\mathcal{G} = \langle S, N, A, P, \overline{R}, \mathcal{L}, \Phi, \gamma\rangle$ *where:*

- $\langle S, N, A, P, \overline{R}, \gamma\rangle$ *is an NMRG;*
- $\mathcal{L} : S \to 2^{\mathcal{AP}}$ *is the labelling function;*
- Φ *is a set of* co-safe LTL *specifications.*

Note that for the NMRG to be consistent, as it can be intuitively inferred, $\overline{R_i} \in \overline{R}$ should be correlated to the progression, satisfaction or violation of any specification $\varphi_k \in \Phi_i$. This correlation must be consistent with the following rule: $r_s > r_p > r_v$.Where Φ_i is the set of specifications associated to agent i, r_s is the reward granted by $\overline{R_i}$ to the agent i when it satisfies any $\varphi_k \in \Phi_i$, r_p is the reward granted when progressing any of those φ_k and r_v is the one granted when violating them. By violation of a specification we mean that it cannot be fulfilled anymore within the current episode.

Henceforth, when we mention NMRGs, we will be referring to NMRGs with LTL specifications. Note that in a NMRG P and $\overline{R}$ are unknown to agents. Again, we cannot apply RL algorithms directly to NMRGs since this would likely lead to non-stationary policies that would not converge. Notice that RL is typically based on Markovian Models; thus, for most of the RL algorithms to work effectively, we need the reward functions to depend solely on the last state and action taken.

3.2 Extended Markov Games

In this work we want to learn general purpose non-Markovian LTL specifications in a multi-agent Markovian setting, so as to exploit MARL techniques. This section is devoted to introduce Extended Markov Games (EMGs) as the mathematical model to achieve this objective.

Definition 6 (Problem statement)**.** *Given a NMRG* $\mathcal{G} = \langle S, N, A, P, \overline{R}, \mathcal{L}, \Phi, \gamma\rangle$ *and* $s_o \in S$ *as the initial state, we want each agent to learn its own non-Markovian optimal policy* $\overline{\pi}^*$ *that for each state chooses the best action leading to satisfy the given* co-safe LTL *specification.*

Further, we introduce a *joint non-markovian policy* $\overline{\Pi}^*$ as the set of the agents' individual (non-markovian) policies $\{\overline{\pi}_1, \ldots, \pi_N\}$. We represent joint actions, i.e, sets of actions simultaneously taken by the agents, as $u \in \mathcal{U}$, where $u = \langle a_1, \ldots, a_N\rangle$ and $\mathcal{U} = A_1 \times \cdots \times A_N$.

Since we need to use a Markovian model to train our MARL agents we need a notion of equivalence between NMRGs and MGs in a similar fashion to what [3] introduced for single-agent models.

Definition 7 (Equivalence)**.** *A NMRG* $\mathcal{G} = \langle S, N, A, P, \overline{R}, \mathcal{L}, \Phi, \gamma\rangle$ *is equivalent to an extended MG* $\mathcal{G}' = \langle S', N, A, P', R', \mathcal{L}, \Phi, \gamma\rangle$ *if there exist two functions* $\tau : S' \to S$ *and* $\rho : S \to S'$ *such that:*

1. *For all* $s \in S$, $\tau(\rho(s)) = s$.
2. *For all* $s_1, s_2 \in S$ *and* $s'_1 \in S'$, *if* $\mathrm{P}(s_1, u, s_2) > 0$ *and* $\tau(s'_1) = s_1$, *there exists a unique* $s'_2 \in S'$ *such that* $\tau(s'_2) = s_2$ *and* $\mathrm{P}(s'_1, u, s'_2) = \mathrm{P}(s_1, u, s_2)$.
3. *For any feasible trajectory* $\langle s_0, u_1, \ldots, s_{l-1}, u_l, s_l\rangle$ *of* $\mathcal{G}$ *and* $\langle s'_0, u_1, \ldots, s'_{l-1}, u_l, s'_l\rangle$ *of* $\mathcal{G}'$, *such that* $\tau(s'_t) = s_t$ *and* $\rho(s_0) = s'_0$, *we have* $\overline{R}(\langle s_0, u_1, \ldots, s_{l-1}, u_l, s_l\rangle) = R'(\langle s'_0, u_1, \ldots, s'_{l-1}, u_l, s'_l\rangle)$.

Where by feasible trajectory *we refer to any trajectory that is consistent within the transitions in the game.*

The crucial points in Def. 7 are clauses (2) and (3), which assert the equivalence of the two models (with respect to the initial states) in both their dynamics and reward structure. In particular, clause (2) ensures that for any given trajectory in $\mathcal{G}$

$$s_0, u_1 \overset{P(s_0,u_1,s_1)}{\longrightarrow} s_1, u_2 \cdots s_{l-1}, u_l \overset{P(s_{l-1},u_l,s_l)}{\longrightarrow} s_l$$

and for any extended state s'_0 with base state s_0, i.e., $\tau(s'_0) = s_0$, there is a trajectory in $\mathcal{G}'$ of similar structure:

$$s'_0, u_1 \overset{P'(s_1,u_1,s_2)}{\longrightarrow} s'_1, u_2 \cdots s'_{l-1}, u_l \overset{P'(s_{l-1},u_l,s_l)}{\longrightarrow} s_l$$

where $\tau(s'_t) = s_t$ for all t in the trajectory. In this case, trajectories $\langle s_0, u_1, \ldots, s_l\rangle$ and $\langle s'_0, u_1, \ldots, s'_l\rangle$ are called *weakly correspondent*. Clause (3) on the other hand, imposes requirements on the individual rewards assigned to each of the agents in the extended game $\mathcal{G}'$. If we have two weakly correspondent trajectories $\langle s_0, u_1, \ldots, s_l\rangle$ and $\langle s'_0, u_1, \ldots, s'_l\rangle$ such that $\rho(s_0) = s'_0$, we say that these trajectories are *strongly correspondent*. This also means that their relationship is one-to-one, that is, each trajectory in $\mathcal{G}$ has a unique strongly correspondent in $\mathcal{G}'$, and that a trajectory in $\mathcal{G}'$ has a unique strongly correspondent if its first state is an initial state. Thus, clause (3) requires that the functions in R' assign individual rewards to extended states in such a manner that strongly corresponding trajectories receive the same reward.

In a similar fashion to what [5] did extending MDPs we introduce the following definition:

Definition 8 (Equivalent MG). *Given an NMRG $\mathcal{G} = \langle S, N, A, P, \overline{R}, \mathcal{L}, \Phi, \gamma \rangle$ with $\Phi = \{\varphi_1, \ldots, \varphi_M\}$, let $\mathcal{A}_k = \langle 2^{\mathcal{AP}}, Q_k, q_{k0}, \delta_k, F_k \rangle$ be the DFA corresponding to φ_k. We define the* equivalent Markov game $\mathcal{G}' = \langle S', N, A, P', R', \mathcal{L}, \Phi, \gamma \rangle$ *such that*

- $S' = Q_1 \times \cdots \times Q_M \times S$ *is the set of states;*
- $\mathcal{U}$, Φ, $\mathcal{L}$ *and* δ *are defined as in the NMRG;*
- $P' : S' \times \mathcal{U} \times S' \to [0, 1]$ *is defined as follows:*

$$\mathrm{P}'\left(\vec{q}, s, u, \vec{q}', s'\right) = \begin{cases} \mathrm{P}\left(s, u, s'\right) & \text{if } \forall i : \delta_i\left(q_i, s'\right) = q_i' \\ 0 & \text{otherwise} \end{cases}$$

- $R' : S' \times \mathcal{U} \times S' \to \mathbb{R}$ *is defined as:*

$$R'\left(\vec{q}, s, u, \vec{q}', s'\right) = \sum_{i : q_i' \in F_i} r_i$$

Intuitively, by Def. 8 the extended state space in $\mathcal{G}'$ is a product of the states in the original NMRG and the automata for the various LTL formulas. Notice that, since we assume a fully observable environment, it is important that the new state space perceived by every agent is composed by all automata. Given a join action u, the s-component in the extended state progresses according to the original dynamics in the NMRG, while the transition function of the corresponding automaton marks the progress of the other components.

Theorem 1. *The NMRG $\mathcal{G} = \langle S, N, A, P, \overline{R}, \mathcal{L}, \Phi, \gamma \rangle$ is equivalent to the MG $\mathcal{G}' = \langle S', N, A, P', R', \mathcal{L}, \Phi, \gamma \rangle$ defined above.*

Given a joint Markovian policy Π' for $\mathcal{G}'$, a policy $\overline{\Pi}$ on $\mathcal{G}$ that guarantees the same rewards can be easily found. To this end, consider a trace $\sigma = \langle s_0, u_1, s_1, \ldots, s_{l-1}, u_l, s_l \rangle$ in $\mathcal{G}$ and let q_i be the state of A_i on the input σ. We define the (non-Markovian) joint policy $\overline{\Pi}$ equivalent to Π' as $\overline{\Pi}(\sigma) = \Pi'\left(q_1, \ldots, q_M, s_l\right)$. Given this and Def. 7, we can define optimal policies for $\mathcal{G}$ by solving $\mathcal{G}'$ instead. Extending [3] to a multi-agent setting we obtain:

Theorem 2. *Given a NMRG $\mathcal{G}$, let Π' be a set of optimal policies for an equivalent MG $\mathcal{G}'$. Then, the policy $\overline{\Pi}$ for $\mathcal{G}$ that is equivalent to Π' is optimal for $\mathcal{G}$.*

It can be then deduced that multi-agent RL techniques can be directly applied to $\mathcal{G}'$, so that we can obtain an optimal joint policy Π'^* for $\mathcal{G}'$. Therefore, an optimal joint policy for the NMRG $\mathcal{G}$ can be learnt from $\mathcal{G}'$.

Remark 1. *Note that none of these game structures is explicitly known by the learning agents, in the sense that the agents are never given the whole structure as input, but rather they just get to know what they can observe through their interactions. Thus, in practice the above transformations are never done explicitly. Instead, the agents learn by assuming that the underlying model is $\mathcal{G}'$.*

Since the state space of $\mathcal{G}'$ is the product of the state spaces of $\mathcal{G}$ and of the automata A_φ, the rewards in R' are Markovian, i.e., the state representation of the temporal formulae in Φ are compiled into the states of $\mathcal{G}'$. As in [8] for single agents, and given Cor. 1, we can state the following result in the multi-agent case:

Corollary 2. *RL for* co-safe *LTL rewards φ over an NMRG $\mathcal{G} = \langle S, N, A, P, \overline{R}, \mathcal{L}, \Phi, \gamma \rangle$ can be reduced to RL over the EMG $\mathcal{G}'$ in Def. 8.*

Remark 2. *We observe that Cor. 2 holds for specifications in Φ expressed in any temporal language as long as there is a way to compute a DFA for each of the specifications.*

This means that we can employ LTL as well as the variants we presented in Section 2, whose queries can be directly transform into DFAs, or even more expressive languages such as Linear Dynamic Logic over finite traces LDL_f. [5].

Example 2. *Consider the specification of making shears we made in our last example. If we want to reward our agents for making shears given the state space we first defined, we would need to check the state-action history of the agents when using the workbench to reward them only if they previously got wood and iron. Since rewards depend on the trace followed by the agents, our model would not be a Markov game anymore but a NMRG. In order to model our problem as an Extended Markov Game, we transform the LTL specification of making shears into a DFA whose states would be given to the agents as an extension of the original observation that they perceive from the environment. The agents would then begin the episodes by perceiving an observation of the map extended by the initial state of the automaton, that represents the whole specification of making shears to be fulfilled. Once the agents have progressed the specification, which in this case means that they got iron or wood, the automaton will transit to a new state that represents the remainder of the specification to be fulfilled. The agents will notice a change in their perception of the environment because of this new state in the DFA. Hence, the agents will perceive the process as Markovian.*

Remark 3. *Note that the dimensionality of the extended state space is not dependant on the number of specifications, as it extends Markov Games only with the representation of the current DFA state. Thus, following the methodology presented in our Section 5, we only need to extend the state space with a single feature. When working with multiple specifications, these can still be represented with a single DFA.*

4 Deep MARL with *co-safe* LTL Goals

To empirically support our theoretical results on EMGs, this section is devoted to introduce two simple decentralized extensions of single-agent Deep Reinforcement Learning (DRL) algorithms designed to work with LTL specifications. The first approach is based on extending with temporal logic specifications a popular baseline in MARL called I-DQN [31], while the second is a multi-agent extension of LPOPL that we referred in Sec 1. The extended algorithms described below are employed in the experiments presented in Sec. 5.

4.1 I-DQN with *co-safe* LTL Goals

The first algorithm is based on DQNs defined in Sec. 2. Q-values for a given agent i can be defined in the multi-agent context [11] by fixing the policy of all the other agents. Let $Q_i^{\pi_i}$ be the Q-value for a given state s and action a for agent i, defined as follows:

$$Q_i^{\pi_i}(s_0, a_i \mid \Pi_{-i}) = \mathbb{E}_{\pi_i}\left[\sum_{t=0}^{T} \gamma^t r_t \mid s_0, a_i, \Pi_{-i}\right] \tag{7}$$

where Π_{-i} are the policies of all agents other then i, and T is the length of the trace.

Intuitively, Equation (7) states that the quality of an action-state pair, given a policy for an agent i, can be expressed as in the single-agent case given the policies of all other agents. This allows Independent Q-learning [33] to train multiple agents in a decentralized fashion. Here we consider a deep learning variant of this algorithm (see, e.g., [31]), where each agent is trained with an independent DQN.

However, in our case, we adopt a decentralized version of an algorithm that uses LTL specifications and LTL progression instead of classical reward functions (see, e.g., [21]). Hereafter we refer to this algorithm as I-DQN-l. Since we are working with sets of specifications, I-DQN-l also incorporates a curriculum learner that selects the specification that the agents will have to fulfill in the next episode. We selected a simple method that assumes the specifications in Φ follow some order $\varphi_0, \ldots, \varphi_{M-1}$ where $|\Phi| = M$, and the curriculum learner just selects the next specification following the given order.

4.2 I-LPOPL

Our second algorithm for experimental evaluations is a multi-agent extension of the LPOPL [35], which is designed specifically to take advantage of *co-safe* LTL specifications in order to boost agents learning performance. Similarly to what we described above for I-DQN-l, we combine the version of LPOPL that employs DQNs as function approximators with Independent Q-learning to obtain a centralized-decentralized algorithm we call Independent-LPOPL (I-LPOPL). Notice that, in MARL, centralized components refer to those that are shared among all the agents, while decentralized are those components that run locally for each agent. Similarly to the single agent algorithm, I-LPOPL is based on four main components:

- **A centralized task extractor.** We use the original task extractor from LPOPL in a centralized manner. This component receives as input the global set of specifications Φ, and returns an extended set Φ^+ of sub-specifications, or *tasks*, which contains each unique LTL task that the original set of specifications can be divided in, including the original set Φ of specifications. For instance, given $\Phi = \{\diamond(b \wedge \diamond c), \diamond(d \wedge \diamond a)\}$, after a first progression the specifications become $\diamond c$ and $\diamond a$. Thus, here the task extractor would generate the extended set $\Phi^+ = \{\diamond(b \wedge \diamond c), \diamond(d \wedge \diamond c), \diamond c, \diamond a\}$.
- **Decentralized LPOPL behavior policies.** In I-LPOPL each agent has its own set of networks, as each agent has a different DQN working as a Q-value function approximator for each extracted task. If φ is the specification to solve in the current episode and φ^+ the result of progressing φ through the history of the episode so far, then the behavior policy is ϵ-greedy on the I-DQN-L whose goal is to satisfy φ^+, i.e, the behavior policy for each agent would be the one optimized to solve the current goal task observed by the agent in the extended state of the EMG.
- **Decentralized Q-value function updates.** In each transition, each agent updates its DQNs independently, that is, at each time step, the sequence $\langle s'_t, u_t, s'_{t+1}, r_{t+1}\rangle$ is stored in the replay buffer of each DQN, where r is given accordingly to the task each DQN is learning. However, in order to allow agents to develop different and potentially complementary individual policies, the update process for each agent is independent from the others'.
- **A centralized curriculum learner**: At the beginning of every iteration of I-LPOPL a global curriculum learning method is used to select the next specification to learn, in the same fashion as in I-DQN-l.

A training iteration of I-LPOPL proceeds as follows: the task extractor receives the set of specifications and creates the sets of tasks. Then, an independent DQN is initialized by each agent for each of these tasks. Once done, the curriculum learner selects the next specification to be solved by the agents in the environment. The agents start interacting with the environment, and once there are enough interactions to start learning, the Q-value function update algorithm is called after each transition. When the curriculum learner triggers the end of the learning process for the current task, a new one is selected. This process is repeated through all specifications.

5 Experimental Evaluation

In this section, we demonstrate that multiple agents can concurrently learn policies that satisfy multiple non-Markovian specifications. Notice that the source code of our experiments is publicly available at `https://github.com/bgLeon/EMG`.

5.1 Experimental Setup

This section is devoted to explain the testing environment, and the features used in our experiments. Our goal here is first to prove that multiple RL agents can be trained to solve multiple non-Markovian specifications, and second that agents can develop multi-agent policies, in this case collaborative policies, to solve the given specifications.

The Minecraft-like world. The environment where we test our algorithms is the Minecraft-like grid map described in Examples 1 and 2. We chose this environment as it is similarly used as testing set-up in [8, 35, 2]. Specifically, our environment and test setting is a multi-agent version of the one presented in [35], where we introduce two agents that need to fulfill different sets of *co-safe* LTL specifications. Two agents cannot be in the same place at the same time, which means that agents can obstruct each other. We model the environment as a discounted reward problem including the relevant DFA. That is, given the current specification φ to be solved and its associated DFA $\mathcal{A}_\varphi$, a reward of -1 is given each time the DFA remains in the same state, 0 if the DFA transitions to a new intermediate state (i.e, the agents have progressed in the specification), and +1 if the DFA reaches a terminal state (i.e, the agents fulfilled the specification).

Features. The two algorithms consider the same features, actions, network architecture, and optimizer. The input features are contained in a vector that also registers the distance of every object to the agent receiving the input, as in [2, 35]. The DQNs code is based on the implementation from OpenAI Baselines [10]. Specifically, we use a feedforward network with 2 hidden layers of 64 neurons using ReLu [12] as activation function. The networks are train using Adam optimization [16] with a learning rate of $5x10^{-4}$, sampling with a batch size of 32 transitions over a replay buffer with a sample size of size 25,000 and updating the target networks every 100 steps. The discount factor for both DQN-l and I-DQN-l is 0.98, which works better with them since their networks are trained to solve a full specification; while the discount factor in both LPOPL and I-LPOPL is 0.9, which is a better choice for these algorithms, since each DQN is focused on solving just a sub-specification or short-term task.

5.2 Specification Sets

This section introduces and explains the different sets of specifications to be solved by the agents, also based on a multi-agent extension of [35, 2]:

- **Specifications as sequences of tasks.** The first set contains 10 specifications from [2] which are a simple sequence of properties to be satisfied by the two agents. For instance, the specification *make_shears* defined in Example 1, can be translated to a sequential LTL specification as $\diamond(\mathit{got\_wood} \wedge \diamond(\mathit{used\_workbench} \wedge \diamond(\mathit{got\_iron} \wedge \diamond \mathit{used\_workbench})))$. While a

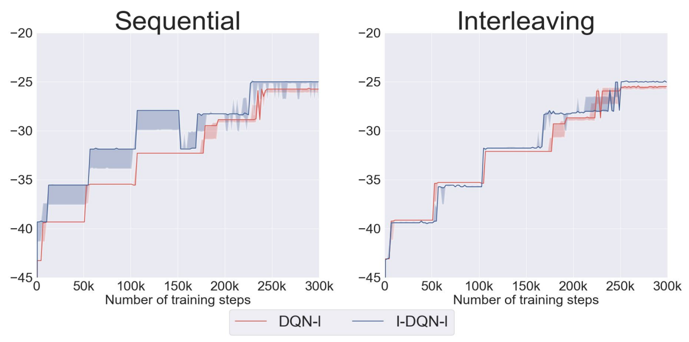

Figure 1. Rewards obtained by DQN-l and I-DQN-l greedy policies evaluated on the whole set of sequence-based specifications (left) and on the whole set of unordered specifications (right). The algorithms were trained for 50k training steps per specification of the set.

single agent can fulfill the whole sequence, an intuitively better policy would be one where one agent goes to get wood and the other waits at the workbench until the first agent collects the wood, then the agent closer to iron moves towards it and the other agent waits at the workbench.

- **Specifications as interleaving tasks.** This set, proposed in [35], differs from the previous one as specifications are no longer a fixed sequence, but rather unnecessary ordering over parts of the specifications are removed. The LTL specification that we used in Example 1 for making shears belongs to this set.

Note that, while we extend the experiments in [35], our single-agent version of the algorithms do not match exactly the setting used in the reference. Specifically, we changed the discount factors, the learning rate, and the reward functions in order to optimize the learning times of the algorithms, a much needed step when doing decentralized multi-agent experiments due to scalability with the number of agents.

5.3 Experimental Results

In order to evaluate our approach, we ran the multi-agent algorithms introduced in Section 4 in a multi-agent version of one of the random maps from [35], and tested them against the correspondent single-agent algorithm in the unaltered map from [35]. In both versions, the agents had a time-limit of 300 steps to solve the specification assigned to an episode. Both algorithms were trained for a given number of training steps per specification (see Figures 1 and 2). Once the limit was reached, the curriculum learner selected the next task to be learnt. Every 1k steps we measured the performance of the greedy policies developed by the algorithms on the whole set of specifications. Figures 1 and 2 shows the mean rewards obtained by these policies, and the shadowed areas are the 25th and 75th percentiles obtained across 3 independent runs per algorithm per set of tasks. Figure 1 compares the performance of I-DQN-l and DQN-l. These algorithms have a different policy network for each specification, where each network is updated only when the curriculum learner have selected the correspondent specification. This is why we observe steeped reward figures in the plot. I-DQN-l achieves higher rewards that DQN-l in the single-agent map thanks to the collaborative strategies that the agents develop in the multi-agent setting, i.e, if the specification requires to collect an item and then use a tool, typically

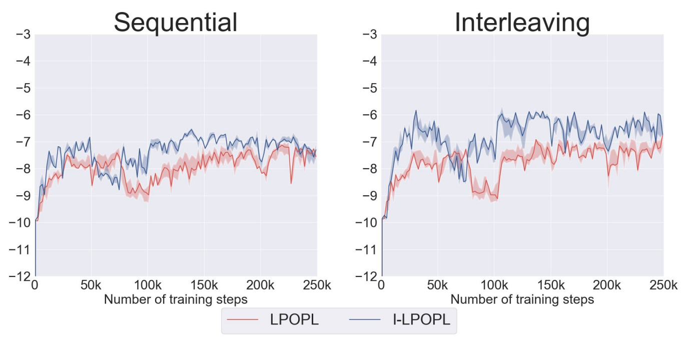

Figure 2. Rewards obtained by LPOPL and I-LPOPL greedy policies evaluated on the whole set of sequence-based specifications (left) and on the whole set of unordered specifications (right). The algorithms were trained for 25k training steps per specification of the set.

one agent goes for the item while the other waits with the tool to use it as soon as the first agent has finished.

Figure 2 show the performance of I-LPOPL and LPOPL. Again, the higher rewards obtained by I-LPOPL over LPOPL are due to agents collaborating in the multi-agent setting. These algorithms do not show steeped reward figures because at each training step every network is updated accordingly to its assigned specification. We note, however, that this policy update system is also causing some drops in the performance of the multi-agent algorithm such as after the 50k step. When the behavior policy is used for a given task, the agents learn that the best policy is for the farthest agent to move towards the next goal in the specification. However, this is not the case for the other policies that are updated without being used as behavior policy. For instance, the agents learn to solve the initial specification first with a collaborative policy since when the two agents going for the same object also means that they are obstructing each other's path. However, when these policy networks are later updated without being used as a behavior policy, the Q-values are updated without encountering the other agent obstruction anymore. Thus, the agent will in some cases forget the collaborative policies and select single-agent ones, causing drops in performance as those in Figure 2.

Finally, we highlight that while I-LPOPL learns faster that I-DQN-l in terms of training steps, I-LPOPL demands higher computing resources and needs longer time to train. The reason for this is that in I-DQN-l we have a DQN per agent per specification, while in I-LPOPL-l we have a DQN per agent per sub-specification. In our experiments, we had 10 specifications in both set of tasks, 27 sub-specifications in the sequential set and 34 in the interleaving set. The table below shows the computing time for each algorithm in the experiments. The experiments were run on a laptop with a GeForce RTX 2080 as GPU, 32 GB of RAM and a i7 7700 HQ as CPU, computing the three independent runs in parallel.

Computing time (hours)		
Algorithm	Sequential	Interleaving
DQN-l	1.20	1.30
I-DQN-l	1.57	2.15
LPOPL	5.13	6.5
I-LPOPL	11.32	14.28

6 Conclusions

We introduced Extended Markov Games as a mathematical model for multi-agent reinforcement learning, to learn policies that satisfy multiple (non-Markovian) LTL specifications in multi-agent systems. Our formal definitions actually infer that any temporal logic can be used to express the specifications as long as they can be converted to a DFA. Our experimental results in collaborative games demonstrate that the proposed framework allows RL agents to learn and develop multi-agent strategies to satisfy different set of specifications.

Based on the present contribution, future directions of research include games with partial observability, where occlusion can refer both to the agent's ability to see the whole map, as well as to the ability to detect other agent's state. As regards the algorithm component, future lines include merging temporal logic with native MARL algorithms that scale better on the number of agents.

REFERENCES

[1] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu, 'Safe reinforcement learning via shielding', in *Thirty-Second AAAI Conference on Artificial Intelligence*, (2018).

[2] Jacob Andreas, Dan Klein, and Sergey Levine, 'Modular multitask reinforcement learning with policy sketches', in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 166–175. JMLR. org, (2017).

[3] Fahiem Bacchus, Craig Boutilier, and Adam Grove, 'Rewarding behaviors', in *Proceedings of the National Conference on Artificial Intelligence*, pp. 1160–1167, (1996).

[4] Fahiem Bacchus and Froduald Kabanza, 'Using temporal logics to express search control knowledge for planning', *Artificial intelligence*, **116**(1-2), 123–191, (2000).

[5] Ronen I Brafman, Giuseppe De Giacomo, and Fabio Patrizi, 'Ltlf/ldlf non-markovian rewards', in *Thirty-Second AAAI Conference on Artificial Intelligence*, (2018).

[6] Yongcan Cao, Wenwu Yu, Wei Ren, and Guanrong Chen, 'An overview of recent progress in the study of distributed multi-agent coordination', *IEEE Transactions on Industrial informatics*, **9**(1), 427–438, (2012).

[7] JCH Christopher, 'Watkins and peter dayan', *Q-Learning. Machine Learning*, **8**(3), 279–292, (1992).

[8] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi, 'Reinforcement learning for ltlf/ldlf goals', *arXiv preprint arXiv:1807.06333*, (2018).

[9] Giuseppe De Giacomo and Moshe Y Vardi, 'Linear temporal logic and linear dynamic logic on finite traces', in *Twenty-Third International Joint Conference on Artificial Intelligence*, (2013).

[10] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

[11] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson, 'Stabilising experience replay for deep multi-agent reinforcement learning', in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1146–1155. JMLR. org, (2017).

[12] Xavier Glorot, Antoine Bordes, and Yoshua Bengio, 'Deep sparse rectifier neural networks', in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, (2011).

[13] Junling Hu, Michael P Wellman, et al., 'Multiagent reinforcement learning: theoretical framework and an algorithm.', in *ICML*, volume 98, pp. 242–250. Citeseer, (1998).

[14] Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann, 'Guided deep reinforcement learning for swarm systems', *arXiv preprint arXiv:1709.06011*, (2017).

[15] Sebastian Junges, Nils Jansen, Christian Dehnert, Ufuk Topcu, and Joost-Pieter Katoen, 'Safety-constrained reinforcement learning for mdps', in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 130–146. Springer, (2016).

[16] Diederik P Kingma and Jimmy Ba, 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980*, (2014).

[17] Orna Kupferman and Moshe Y Vardi, 'Model checking of safety properties', *Formal Methods in System Design*, **19**(3), 291–314, (2001).

[18] Bruno Lacerda, David Parker, and Nick Hawes, 'Optimal policy generation for partially satisfiable co-safe ltl specifications', in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, (2015).

[19] Borja G. León and Francesco Belardinelli, 'Extended markov games to learn multiple tasks in multi-agent reinforcement learning', *arXiv preprint arXiv:2002.06000*, (2020).

[20] Michael L Littman, 'Markov games as a framework for multi-agent reinforcement learning', in *Machine learning proceedings 1994*, 157–163, Elsevier, (1994).

[21] Michael L Littman, Ufuk Topcu, Jie Fu, Charles Isbell, Min Wen, and James MacGlashan, 'Environment-independent task specifications via gltl', *arXiv preprint arXiv:1704.04341*, (2017).

[22] George Rupert Mason, Radu Constantin Calinescu, Daniel Kudenko, and Alec Banks, 'Assured reinforcement learning with formally verified abstract policies', in *9th International Conference on Agents and Artificial Intelligence (ICAART)*. York, (2017).

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al., 'Human-level control through deep reinforcement learning', *Nature*, **518**(7540), 529, (2015).

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

[25] Andrew Y Ng, Daishi Harada, and Stuart Russell, 'Policy invariance under reward transformations: Theory and application to reward shaping', in *ICML*, volume 99, pp. 278–287, (1999).

[26] Amir Pnueli, 'The temporal logic of programs', in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pp. 46–57. IEEE, (1977).

[27] Michael O Rabin and Dana Scott, 'Finite automata and their decision problems', *IBM journal of research and development*, **3**(2), 114–125, (1959).

[28] 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).

[29] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov, 'Proximal policy optimization algorithms', *arXiv preprint arXiv:1707.06347*, (2017).

[30] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua, 'Safe, multi-agent, reinforcement learning for autonomous driving', *arXiv preprint arXiv:1610.03295*, (2016).

[31] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente, 'Multiagent cooperation and competition with deep reinforcement learning', *arXiv preprint arXiv:1511.08779*, (2015).

[32] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente, 'Multiagent cooperation and competition with deep reinforcement learning', *PloS one*, **12**(4), e0172395, (2017).

[33] Ming Tan, 'Multi-agent reinforcement learning: Independent vs. cooperative agents', in *Proceedings of the tenth international conference on machine learning*, pp. 330–337, (1993).

[34] 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).

[35] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith, 'Teaching multiple tasks to an rl agent using ltl', in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 452–461. International Foundation for Autonomous Agents and Multiagent Systems, (2018).

[36] Min Wen, Ivan Papusha, and Ufuk Topcu, 'Learning from demonstrations with high-level side information', in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, (2017).

[37] Dayong Ye, Minjie Zhang, and Yun Yang, 'A multi-agent framework for packet routing in wireless sensor networks', *sensors*, **15**(5), 10026–10047, (2015).

[38] Weichao Zhou and Wenchao Li, 'Safety-aware apprenticeship learning', in *International Conference on Computer Aided Verification*, pp. 662–680. Springer, (2018).