On Tackling Complex Tasks with Reward Machines and Signal Temporal Logics

Ana María Gómez Ruiz, Thao Dang, Alexandre Donzé

On Tackling Complex Tasks with Reward Machines and Signal Temporal Logics

Ana María Gómez Ruiz[1], Thao Dang[2] and Alexandre Donzé[1]

[1]VERIMAG, Université Grenoble Alpes {firstname.lastname}@univ-grenoble-alpes.fr
[2]CNRS, VERIMAG, Université Grenoble Alpes {firstname.lastname}@univ-grenoble-alpes.fr

Abstract— **We propose a Reinforcement Learning (RL) based control design framework for handling complex tasks. The approach extends the concept of Reward Machines (RM) with Signal Temporal Logic (STL) formulas that can be used for event generation. The use of STL allows not only a more efficient representation of rewards for complex tasks but also guiding the training process to converge towards behaviors satisfying specified requirements. We also propose an implementation of the framework that leverages the STL online monitoring algorithms. We illustrate the framework with three case studies (minigrid, cart-pole and high-way environments) with non-trivial tasks.**

I. INTRODUCTION

RL has recently been proven to be efficient for solving decision-making problems in various domains, including classical control tasks, games, and multi-agent systems. A key element in RL is specifying a reward function that encodes the goal. However, this concept of reward is difficult to extend to complex task specifications, for example, a robot should reach a number of target points in a given order without entering some unsafe area. Such history-dependent specifications can be, on the other hand, described conveniently using temporal logic.

Recently there has been significant research interest on using temporal logics for specifying RL tasks [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13]. Most of this work focuses on transforming specifications into a specification automaton, which is then composed with the Markov Decision Process (MDP) representing the environment to be controlled. Then the rewards are defined for the transitions of the resulting MDP, using reachability probabilities, to capture the acceptance conditions of the specification automaton. The theoretical questions on the optimality preservation of the transformation from Linear Temporal Logic (LTL) specifications to reward-based specifications are addressed in [14]. A more direct approach [15] is to define an automata-based model called RM to provide a modular and hierarchical framework for defining rewards in complex tasks. This approach can handle non-Markovian reward functions that depend not only on the current state but also on the evolution history of the environment. The authors also propose a Q-learning based algorithm, called QRM, to learn control policies. A later paper [16] proposes variants of QRM including a hierarchical RL algorithm (HRM) to learn policies for tasks specified using RM. In [17] the authors demonstrate how to transform LTL specification into RM, while [18] proposes a notion of RM decomposition to achieve decentralized learning in a cooperative multi-agent context. Moving to continuous-time settings, in [19], a multi-agent deep Q-learning algorithm is proposed to learn the optimal policies (in a finite-state multi-agent stochastic game framework) that maximize the expected robustness degree of the Signal Temporal Logic (STL) specification describing mission objectives, against the best responses of the adversarial agents. To handle non-Markovian STL specifications, [20] proposes τ-dMDP, an extended MDP using past system states and control actions. Using the τ-dMDP, a Deep RL algorithm is proposed to design a networked controller subject to network delays to complete an STL specification. In [1], two problems of learning optimal policies for STL specifications are formulated as maximizing respectively the probability of satisfaction and the expected robustness degree. On this field of formal specification guided RL, the reader is also referred to a recent survey [21].

While the non-Markovian reward issue is resolved, defining rewards to reflect complex control objectives and to enable efficient policy learning is still a challenge. Indeed, conventional learning algorithms often become trapped in local optima, or, when reward information is too sparse, may spend many iterations exploring experiences that do not meaningfully contribute to optimizing the reward. The goal of this paper is to address these issues by proposing an extension of the RM concept using STL. The motivation for this extension is twofold. First, STL being defined over real-valued signals, enables a more expressive and compact representation of history-dependent reward mechanisms. It also offers a principled way to characterize behaviors that are trapped near local optima, allowing the training process to identify and circumvent such suboptimal convergence. Second, while the concepts of RM and TL in discrete time have been extensively studied —with many theoretical aspects such as convergence and optimality already addressed— most existing methods rely on transforming TL into RM. This transformation typically produces large automata, which can obscure the interpretability of the original task specification. We propose a framework that combines RM with STL where STL, well-suited for interpretable behavioral descriptions, is used directly for event generation, while RM serve to define task goals in a modular and operationally effective manner. Furthermore, we leverage efficient methods for online monitoring of STL specifications implemented in the RLRom framework [1] for training models that perform complex tasks or reliably converge toward behaviors satisfying specified requirements.

Our contributions can be summarized as follows:

- We introduce a new framework, called RM-STL, for RM incorporating STL predicates. The framework allows multiple RM to be composed in parallel, where the overall reward is computed as a weighted sum of the individual RM rewards.
- We propose an implementation of the framework, which has been successfully applied for a number of case studies.

The paper is organized as follows. We first review the problem of RL with RM. Then we describe our contributions: an extension of RM with STL predicates, and a learning framework for such RM. We finally describe our implementation and demonstrate the approach on a minigrid, a cart-pole and highway-env examples with non-trivial tasks.

[1] `https://github.com/decyphir/rlrom`

II. Preliminaries

A. Reinforcement learning problem

The RL problem is formalized using an MDP in form of a tuple $M = (S, A, r, p, \gamma)$ where S is a set of environment states, A is a finite set of actions, $r : S \times A \times S \to \mathbb{R}$ is a reward function, $p(s_{t+1}|s_t, a_t)$ is a transition probability distribution, and $\gamma \in (0, 1]$ is a discount factor. Some states can be labeled as *terminal*. The learning process is organized in episodes. One episode is defined as a sequence of interactions between an agent and its environment that starts from an initial state and ends when a terminal state is reached or a stopping condition is satisfied. At each time step t of an episode, observing that the environment is at a state $s_t \in S$, the agent selects and executes an action according to a probability distribution over the actions $\pi(\cdot \mid s_t)$, called a policy. We assume in this work that the agent can observe the full state of the environment. The environment moves to a new state s_{t+1} according to $p(\cdot \mid s_t, a_t)$ and the agent receives a reward $r(s_t, a_t, s_{t+1})$. The objective of the agent is to find a policy π^* to maximize the expected discounted future reward from any state in S. Every optimal policy π^* is known to satisfy the Bellman equations for every state $s \in S$ and every action $a \in A$: $q^{\pi^*}(s,a) = \sum_{s' \in S} p(s' \mid s,a) r(s,a,s') + \gamma \max_{a' \in A} q^{\pi^*}(s',a')$, where $q^{\pi}(s,a)$ is the q-function defined as the expected discounted future reward for taking the action a from the state s and then following the policy π.

B. Reward machines

The reward function in the RL framework is a way to reflect the tasks that the agent is expected to perform. However, being defined only over the states and actions, reward functions can not capture complex non-Markovian tasks which are history dependent. This motivated the introduction of RM which are based on finite state machines [22]. As an agent interacts with the environment moving from state to state, it also transitions between states within an RM, which is determined by high-level events from the environment that the agent can detect. To specify such events, it is often assumed that a set of propositions P and a labeling function $L : S \times A \times S \to 2^P$ are given. For an experience (s,a,s') (where s' is the resulting state after executing the action a from the state s), the labeling function assigns truth values to the propositional symbols in P. These truth assignments are inputs to the RM.

Definition 1 (RM): Given a set of propositional symbols P, a set of environment states S, and a set of actions A, a RM is a tuple $RM_{P,S,A} = \langle U, u_0, \delta_u, \delta_r \rangle$ where U is a finite set of RM states, $u_0 \in U$ is an initial RM state, $\delta_u : U \times 2^P \to U$ is the RM state-transition function, and $\delta_r : U \times U \to [S \times A \times S \to \mathbb{R}]$ is the transition-reward function.

The RM $\mathsf{R}_{P,S,A}$ starts in the initial RM state u_0. At every step t, the RM receives as input a truth assignment σ_t which contains all the propositions in P that are true in (s_t, a, s_{t+1}). The rewards are Markovian with respect to $(s_t, u_t) \in S \times U$; however, the RM state u_t at time step t depends not only on the current environment state s_t but also on the history of the environment evolution. Hence, using such a RM one can specify non-Markovian reward functions [15]. Some RM states can also be labeled terminal.

Furthermore, as we will see, in order to incorporate STL predicates as RM propositions, we need to keep the history of the environment evolution. To this end, we define an experience sequence of length $t > 0$, denoted by $\xi_t = s_1, a_1, \ldots, a_{t-1}, s_t$ where s_1 is an initial state of the environment. Let Ξ be the sets of experience sequences that the environment can perform. For a given set of propositions P, we extend the definition of labeling function to $L : \Xi \to 2^P$.

This following definition is for MDP coupled with multiple RM.

Definition 2 (MDP-RM): Given an MDP $\mathsf{M} = (S, A, p, \gamma)$, a set of propositional symbols P, a labeling function L, and a set of RM $\{\mathsf{R}^1, \ldots, \mathsf{R}^m\}$ where $\mathsf{R}^i_{P,S,A} = \langle U, u_0, \delta^i_u, \delta^i_r \rangle$ with $i \in \{1, \ldots, m\}$, we define $\mathsf{M}_{RMs} = (S, A, p, \gamma, P, L, U, u_0, \{\sigma^1_u, \ldots, \sigma^m_u\}, \{\sigma^1_r, \ldots, \sigma^m_r\})$.

The execution of the environment coupled with the RM and the associated reward are defined as follows. At a time step $t > 0$,

- an action $a \in A$ is computed based on some policy depending on the aggregated state $\mathbf{s} = (s, \mathbf{u})$ where s is the current environment state, and $\mathbf{u} = (u_1, \ldots, u_m)$ is the vector of the current RM states;
- this action a leads the environment to the next environment state s', as defined by the MDP;
- the experience sequence is updated by concatenating it with the new action and state: $\xi = \xi \cdot (a, s')$;
- based on the labeling function over the updated experience sequence, the next state u'_i for each RM R^i is determined: $u'_i = \delta^i_u(u_i, L(\xi))$;
- the overall reward associated with this step is then defined as:

$$\mathscr{R}_t = \sum_{i \in 1, \ldots, m} w_i \delta^i_r(u_i, u'_i)$$

where $w_i \in \mathbb{R}$ is a weight associated with the RM R^i.

The policy of selecting actions is part of the chosen training algorithm. Note that any standard RL method is supported by our framework. Indeed, the aggregate state of the environment and the RM state is used to define the observation space. In this way, the value functions computed by the training algorithms can explicitly account for the reward mechanisms captured by the RM. We remark that in previous work on RL with RM [15], separate value functions are typically computed for each RM state. By contrast, using the aggregate state representation allows for a more compact and unified encoding of the value functions. However comparing generic vs specific training methods for RL with RM is beyond the scope of this paper.

III. Reward machines with STL (RM-STL)

A. STL specifications

STL is a logic typically interpreted over dense-time signals that take values in a continuous metric space [23]. STL uses signal predicates μ over a signal y of the form $f(y(t)) \sim 0$

where f is a scalar-valued function and $\sim \in \{<, \leq, >, \geq, =, \neq\}$. It is then defined recursively using Boolean combinations of subformulas, or by applying an interval-restricted temporal operator (such as "always" ($\Box$), "eventually" ($\Diamond$), and "until" ($\mathscr{U}$) which have the usual meaning) to a subformula. If I is a time interval, the syntax for STL is defined as: $\varphi := \top \mid \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathscr{U}_I \varphi_2$. The "eventually" $\Diamond$ operator is defined as $\Diamond_I \varphi \triangleq \top \mathscr{U}_I \varphi$, and the "always" $\Box$ operator is defined as $\Box_I \varphi \triangleq \neg(\Diamond_I \neg\varphi)$. The semantics are described informally as follows. The signal y satisfies $f(y) > 0$ at time t if $f(y(t)) > 0$. It satisfies $\varphi = \Box_{(0,1]}(f(y) = 0)$ if for all time $0 < t \leq 1$, $f(y(t)) = 0$. The signal satisfies $\varphi = \Diamond_{[1,2)}(f(y) < 0)$ iff there exists a time t such that $1 \leq t < 2$ and $f(y(t)) < 0$.

a) Quantitative semantics: We use the quantitative semantics of STL [24] defined as robustness of satisfaction, to provide a fine-grained exploration. Given a signal y and an STL formula φ, a function ρ is defined such that when $\rho(\varphi, y, t)$ is positive it indicates that (y, t) satisfies φ, and its absolute value estimates the *robustness* of this satisfaction. If φ is of the form $f(y) > b$, then its robustness is $\rho(\varphi, y, t) = f(y(t)) - b$. For the conjunction $\varphi := \varphi_1 \wedge \varphi_2$, we have $\rho(\varphi, y, t) = \min(\rho(\varphi_1, y, t), \rho(\varphi_2, y, t))$, while for the disjunction $\varphi := \varphi_1 \vee \varphi_2$, $\rho(\varphi, y, t) = \max(\rho(\varphi_1, y, t), \rho(\varphi_2, y, t))$. For a formula with until operator $\varphi := \varphi_1 \mathscr{U}_I \varphi_2$, $\rho(\varphi, y, t) = \max_{t' \in I}\left(\min\left(\rho(\varphi_2, y, t'), \min_{t'' \in [t, t']}(\rho(\varphi_1, y, t''))\right)\right)$.

STL satisfaction robustness is defined for signals over some given horizon. In order to use it in the RL framework where we need to deal with partial signal representing the evolution of the environment and the RM before the end of an episode, we use the notion of robustness interval, introduced in [25] for online monitoring purposes, to enclose the exact robustness value that cannot yet be computed since the remaining portion of the signal is unknown. In [25] an algorithm is proposed to compute the robustness interval $[\underline{\rho}, \overline{\rho}]$ for a given STL formula φ and a given partial signal.

In a standard reinforcement learning setting, an agent interacts with its environment at discrete time steps, consequently, signals are observed in discrete time. We use an interpolation to obtain continuous-time signals. For simplicity, we use the same notation to mean both a signal in discrete time and its associated continuous-time signal.

B. Using STL with RM

We use the RM structure from Definition 2, while enriching the set P of propositions with formulas involving STL robustness of the form $\underline{\rho}(\varphi, y_{[0,\tau]}, t) \geq \beta$ or $\overline{\rho}(\varphi, y_{[0,\tau]}, t) \leq \beta$ where $t \leq \tau$ and φ is an STL formula. Here, $y_{[0,\tau]}$ represents the evolution of the observation from the initial step to step τ. It is possible that the formulas involve only a subset of variables in the observation. We refer to the STL formulas used to define RM transitions as *event formulas*.

The resulting MDP-RM works as follows. At a time step t, let $\xi_t = s_0, a_1, s_1, \ldots, s_{t-1}, a_{t-1}$ be the sequence of experiences of the environment up to time step $(t-1)$. Let $\mathbf{u}_{t-1}$ be the vector of the current states of the RM. For each proposition φ in P that involves STL robustness, to determine its truth value we evaluate the interval of values for each robustness variable in the formula. Let σ_t be a set of all the formulas in P that are true at time step t. The truth assignment σ_t is then received by the RM to decide the transitions to take from the current RM states $\mathbf{u}_t$.

Besides event formulas, STL can be used to specify non-functional properties, which we refer to as *evaluation formulas*. These formulas are assessed over complete episodes to measure the agent's performance. A straightforward approach might incorporate the online robustness of evaluation formulas into the observation space or use it to shape the reward signal as in [26]. However, because evaluation formulas typically span long horizons—designed to evaluate complete episodes—their instantaneous robustness values offer limited insight into overall satisfaction, which can only be determined at the end of each episode. This poses a challenge for preserving the Markov property of the reward. For precisely this reason, it is also preferable to use short-horizon event formulas, where online robustness is more informative, and in this case the observation of the training algorithm can be additionally augmented with the robustness variables to encode in the value function the information about important events. These short-horizon formulas are assessed in a sliding-window fashion If hz is the horizon of the formula, this means that we use as observation $\rho(\varphi, y, t^{-hz})$ where $t^{-hz} = \max(0, t - hz)$.

Discussion: Before continuing, we briefly discuss the advantage of the proposed combination RM-STL. The motivation for this extension, as mentioned earlier, is to directly capture events that depend on the evolution of the environment and RM states. These events are significant because they relate either to the reward mechanism or to the training guidance. Such guidance can steer the training algorithm toward exploring behaviors that are promising for optimizing the expected future reward, or prevent it from persisting in blocking behaviors that arise when the environment becomes trapped near a local optimum—situations that can be effectively characterized using STL. In these cases, the stopping condition can be encoded as a transition leading to a terminal state. These uses of STL with RM will be illustrated in the case studies.

Another advantage of this combination is interpretability. Using STL predicates to define transitions of RM and monitoring their robust satisfaction during the training process provide an expressive and yet interpretable means of modeling sophisticated reward mechanisms that can differentiate among subtle behavioral contexts.

IV. EXPERIMENTAL RESULTS

To evaluate RM-STL on complex tasks within RL environments, we present three distinct environments as case studies: Minigrid, Cart-pole and Highway-env. In this section, we first describe the implementation, then each environment and the design of their specific tasks, and finally, the experimental results.

A. Implementation

The implementation of our framework RM-STL builds upon the RLRom framework[2], which can evaluate RL agents through interpretable monitors. RLRom receives STL predicates as inputs, which are used to monitor the behavior of the agent and guide the training process. Within this framework, the environment is wrapped sequentially in two layers:

1) an STL wrapper that evaluates the truth value and robustness of the user-defined STL predicates at each step of the episode. This wrapper serves as a real-time logical monitor, computing satisfaction degrees that reflect how well the behavior of the agent aligns with temporal and logical constraints (*e.g.*, "the agent must pick up the key before unlocking the door", or "the car must keep a safe distance from the closest car"). The STL robustness evaluation over partial signals is based on STLRom[3] which provides online robust monitoring as described in [25].
2) an RM wrapper that constructs the reward signal according to the RM transitions by augmenting the observation space with the RM states and STL robustness, producing rewards consistent with the RM structure.

The two wrappers operate hierarchically, first applying logical monitoring via STL then the RM to structure task execution. This hierarchical design results in a flexible and interpretable framework: flexible, because STL predicates can be easily redefined or combined to express new temporal or logical constrains without modifying the underlying environment or learning algorithm; and interpretable, because the RM explicitly encodes the progress of the agent through discrete task stages, making it possible to trace why rewards are assigned and how decisions relate to high-level objectives. Together, these properties enable the same framework to generalize across multiple environments while maintaining a clear correspondence between agent behavior and task logic.

Concerning the RL environments in the case studies, they were implemented using Python Gymnasium libraries [27] that provide a standardized API for defining and interacting with environments. All agents were trained using the Proximal Policy Optimization (PPO) algorithm implemented in Stable-baselines3 [28] (SB3), though RLRom makes it possible to pick any algorithm from SB3. The hyperparameters used in this implementation are in Table III in the Appendix.

B. Minigrid

The first case study uses Minigrid library, which provides a collection of 2D grid-world environments with goal-oriented tasks [29]. We used *Unlockenv* as a base environment, but its grid size was increased to test scalability, as illustrated in Figure 1. The objective of the agent is to pick up the key and then open the door. The agent in these environments is a triangle-like agent with a discrete action space, which consists of seven actions (0–6), enabling the agent to move

[2]https://github.com/decyphir/rlrom
[3]https://github.com/decyphir/stlrom

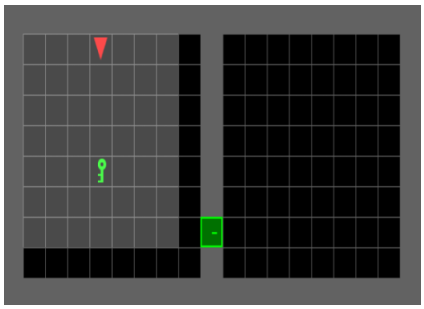


Fig. 1: The minigrid environment *Unlockenv* with bigger grid size.

(left, right, forward), to interact with objects (*e.g.*, key, lava, ball), to perform pickup and drop actions, and to use the toggle action to activate objects, such as opening a door when positioned in front of it. The reward of $1-0.9\frac{n}{n_{max}}$ is given for success, and 0 for failure (where n is the number of steps until success, and n_{max} is the maximal number of steps of an episode defined as a function of the grid size). The episode ends if the agent opens the door or timeout, that is n_{max} steps are reached. The observation space of the environment is partial and egocentric. It is represented as a 7×7 grid centered around the current position and orientation of the agent. Each cell in the grid is described by a 3-dimensional tuple: (*object_idx, color_idx, state*), where *object_idx* and *color_idx* correspond to the type and color of the objects observed by the agent, respectively. The *state* component indicates the status of the object (*e.g.*, door open or closed). The objects included in this environment are the agent, a key, a door and surrounding walls that define the grid boundaries.

The objective of the agent can be decomposed into two sequential tasks: first, picking up the key, and then opening the door. Figure 2 illustrates the corresponding RM, which structures the task into three states. The transitions between these states are governed by STL specifications, defined as $\varphi_1 := \Diamond_{[0,T]} has\_key$ and $\varphi_2 := \Diamond_{[0,T]} has\_key \wedge open\_door$. The value next to the specification corresponds to the reward associated with taking that transition. In this machine, u_0 denotes the initial state (before obtaining the key), u_1 represents the intermediate state where the agent holds the key, and u_2 corresponds to the terminal state, achieved when the agent successfully opens the door using the key.

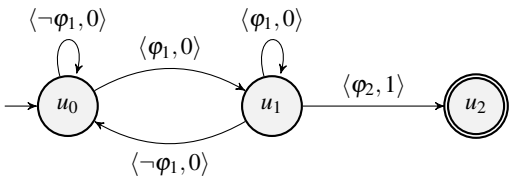
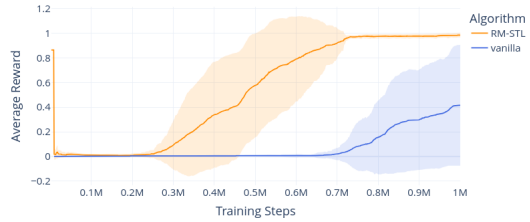
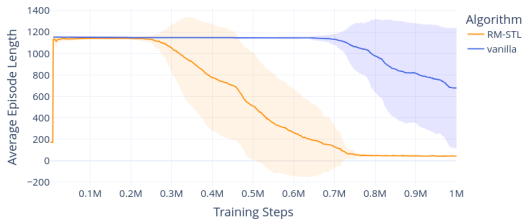


Fig. 2: Reward machine of the Minigrid agent

The training was conducted on environments of increasing complexity, starting with grid size 6x6 and progressively expanding up to 12×12. For each grid configuration, two



(a) Average episode reward.



(b) Average episode length.

Fig. 3: Results from training an agent in the *Unlock* environment from a extended grid size 12x12 in MiniGrid. Blue: vanilla; orange: RM state in observation with STL specification during training.

models were trained: a baseline vanilla PPO agent and the proposed RM-STL implementation, under identical hyperparameters 5 times each. Figure 3 corresponds to the results of the largest grid, illustrating the most challenging scenario. These results include two key performance metrics, episode reward vs number of training time steps, and episode length vs number of training time steps, allowing a comprehensive comparison of both approaches across different environment complexities.

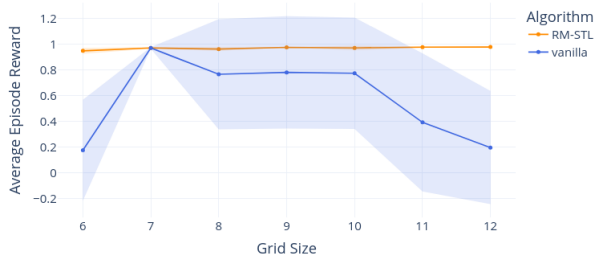
The plot in Figure 4, where the x-axis represents the grid size and the y-axis the corresponding averaged metrics, demonstrates that the RM-STL agent consistently outperforms the vanilla PPO as the environment complexity increases, achieving higher rewards and shorter episode lengths. More details of the results are shown in Table I. This trend highlights the improved learning efficiency and generalization capabilities provided by integrating STL specifications and RM into the training process.

C. Cart-Pole

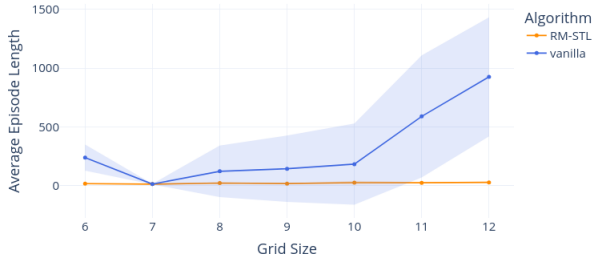
The second case study involves a cart-pole environment, where a pole is attached by an unactuated joint to a cart which moves along a friction-less track. The pole is placed upright on the cart and the goal is to balance the pole by applying forces in the left and right direction on the cart. The default reward is 1 for every step taken if the pole is upright. The state space is continuous and has 4 dimensions (cart position, cart speed, pole angle, pole angular speed). The agent has 2 available discrete actions: push the cart to the right or to the left. The episode ends if the absolute value of pole angle is greater than 12^o, or the absolute value of

TABLE I: Performance comparison per grid size of RM+STL vs Vanilla

Grid Size	Algorithm	Mean Reward ± Std	Mean Len Episode ± Std
6	Specs	0.9494 ± 0.0221	16.2 ± 7.09
6	Vanilla	0.1756 ± 0.3927	238.2 ± 111.36
7	Specs	0.9720 ± 0.0062	12.2 ± 2.68
7	Vanilla	0.9715 ± 0.0060	12.4 ± 2.61
8	Specs	0.9631 ± 0.0128	21.0 ± 7.28
8	Vanilla	0.7670 ± 0.4290	121.2 ± 218.63
9	Specs	0.9767 ± 0.0015	16.8 ± 1.10
9	Vanilla	0.7817 ± 0.4370	142.8 ± 282.42
10	Specs	0.9721 ± 0.0171	24.8 ± 15.24
10	Vanilla	0.7748 ± 0.4335	182.4 ± 345.66
11	Specs	0.9782 ± 0.0050	23.4 ± 5.41
11	Vanilla	0.3926 ± 0.5375	588.8 ± 519.24
12	Specs	0.9789 ± 0.0055	27.0 ± 7.00
12	Vanilla	0.1969 ± 0.4402	925.6 ± 506.25



(a) Average episode reward.



(b) Average episode length.

Fig. 4: Results achieved by the trained models when evaluated on larger grid sizes.

the cart position is greater than 2.4, or if the episode length is greater than 500 time steps [30].

To make more complex tasks in this environment, specific target positions are defined. Figure 5 illustrates the initial position of the cart-pole system, along with the designated target regions. The first objective of the cart is to reach the region A, where the x position must be between -0.7 and -0.5 (formally: $\mu_A = -0.7 < x < -0.5$). Subsequently, the cart has to move to the region B, located between 0.5 and 0.7 (formally: $\mu_B = 0.5 < x < 0.7$), all while keeping the pole upright throughout the episode. This task is significantly more difficult than the original cart-pole task because in addition to not falling, the agent has to perform a sequence of subtasks (move left, move right, and then brake). It is not easy to use simple reward functions to specifying such a task

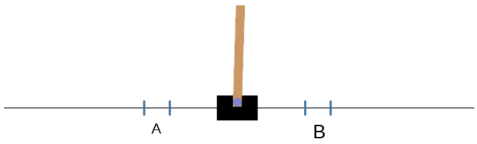


Fig. 5: The cart-pole environment, augmented with target regions A and B defined as STL predicates μ_A and μ_B.

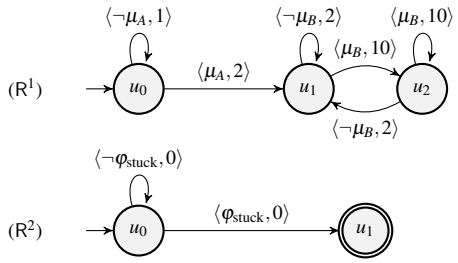


Fig. 6: Reward machines of the cart pole: R^1 encodes the task to go to region A and then to region B and stay there. R^2 enforces episode termination when the formula φ_{stuck} defined in formula (1) is true.

composed of three subsequential subtasks, since the reward should reflect which subtasks have been achieved and which remain to be executed. The RM describing this objective is presented in Figure 6, where the goal is to reach the state u_2 and stay there for the length of the episode. To keep the incentive to not fall, a minimum reward of 1 is provided for each step, as in the original reward formulation. This reward is doubled when the cart-pole has reached the region A, and then the reward becomes 10 for each step whenever the cart-pole is in B.

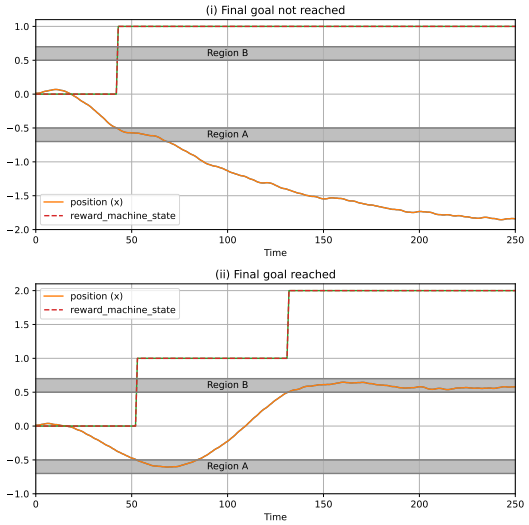


Fig. 7: Test results of a single episode using the sub-optimal trained model (i) and using a successfully trained one (ii).

Figure 6 also depicts another RM for the cart-pole which encodes a training guidance. Indeed, after reaching the region A, the agent may not move to B because it can still get a reward of 2 at each step. In order to encourage the exploration towards B, we describe, by the STL formula (1), situations where at some time point (in this example, between the 100^{th} step and the 200^{th} step) the cart continuously stays in the region with $x < 0$ for some time (in this example, for 300 successive steps).

$$\varphi_{\text{stuck}} = \Diamond_{[100,200]} \Box_{[0,300]} (x < 0) \tag{1}$$

Using this in a RM with a terminal state penalizes the sub-optimal policy illustrated on Figure 7 (i). Figure 8 shows the results of 10 instances training first without R^2, then with R^2. We show the mean average cumulated reward, as well as the average cumulated reward of the worst and the best policies obtained both without R^2 and with R^2. For the latter case, *i.e.*, using R^2, we see that the sub-optimal policy is completely eliminated, as the average cumulated reward eventually reaches values around 3.000 for all training instances.

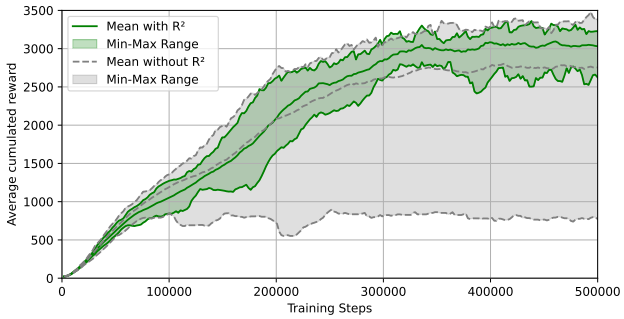


Fig. 8: Results for 10 training instances with (green) and without (gray) R^2 encoding STL formula φ_{stuck} as terminal condition. With R^2, the performance of the policy consistently reaches an accumulated reward of 3.000, indicating that the task is successfully completed, whereas with only R^1, the performance of some policies remains around 800, indicating that only the region A is reached in most episodes.

D. Highway-env

Highway-env is a minimalist environment for decision-making in autonomous driving. It includes basic driving scenarios such as highways, intersections, roundabouts, race-tracks, and parking setups, where the agent can perform different tactical decision making tasks in the RL context. The vehicles populating the road follow simple and realistic behaviors that dictate how they accelerate and steer on the road. For this experiment, we use an environment where the observation has the position x^{ego} and y^{ego}, the speed v_x^{ego} and v_y^{ego} of the ego vehicle, as well as position and speed for the four closest vehicles in front of it, denoted as x^i, y^i, v_x^i and v_y^i for $i \in \{1,2,3,4\}$. The available actions of the agent are: turn left or right, idle, go faster or slower. The original reward function consists of a velocity term and a collision term. In our experiment, we aim at training agents to not only drive fast and safe, but also to follow certain driving behaviors. For this, we designed a set of formulas, described in Table II

Formula	hz	Interpretation
$\mu^i_{danger} = \lvert x^i\rvert < .1 \wedge \lvert y^i\rvert < .1$	0	car i is in front and too close
$\mu_{danger} = \vee_{i=1}^{4}\ \mu^i_{danger}$	0	A car is in front and too close
$\varphi_{tail} = \square_{[0,10]}\ \mu_{danger}$	10	Too close for too long
$\mu_{fast} = v_x^{ego} > 25$	0	ego vehicle goes fast
$\varphi_{lazy} = \square_{[0,10]}\ \neg\mu_{fast}$	10	safe for 10 steps
$\varphi_{fast} = \Diamond_{[0,85]}\square_{[0,15]}\ \mu_{fast}$	100	always accelerates eventually
$\mu_{right} = y^{ego} > 0.6$	0	ego is in right lane
$\varphi_{left} = \square_{[0,10]}\ \neg\mu_{right}$	10	not in right lane for 10 steps
$\varphi_{right} = \Diamond_{[0,85]}\square_{[0,15]}\ \mu_{right}$	100	always in right lane eventually

TABLE II: Formulas characterizing driving behaviors and used in the RM in Figure 9. Short horizon formula ($hz \leq 10$) are used as event formulas. Long horizon formulas ($hz = 100$) are for agent evaluation past training.

and RM described in Figure 9. They are designed to enforce three main goals: 1) avoid collision, 2) go fast and 3) prefer the right lane. These goals are often in competition with one another, but with our framework, we can balance them in a comprehensive way. For instance, high velocity should not be rewarded when the ego vehicle is close to a collision with another car, *etc*. By tuning the rewards and penalties associated with each transition, we can also influence which policy is prioritized wrt to one another. For illustration, in Fig 10, we show the evolution of formula satisfaction and metrics during training (testing and monitoring 100 episodes every 1e5 steps of training), which is consistent with the definitions in Table II and Fig 9.

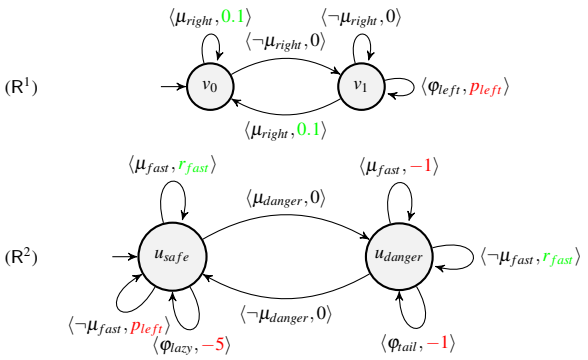


Fig. 9: Reward machine and formulas for the highway-env environment.

V. Conclusion

In this paper, we proposed an extension of RM using STL predicates to incorporate strategies to guide RL. This framework accelerates the training process by avoiding experiences that maintain the agent around local optima or are not robust with respect to safety specifications. We implemented our algorithms and demonstrated their effectiveness across three case studies.

Future work includes using STL for reward shaping. Similar to the essential ideas of the potential-based reward shaping approach [16], but instead of treating RM as MDP, STL can be used to discover promising or, on the contrary,

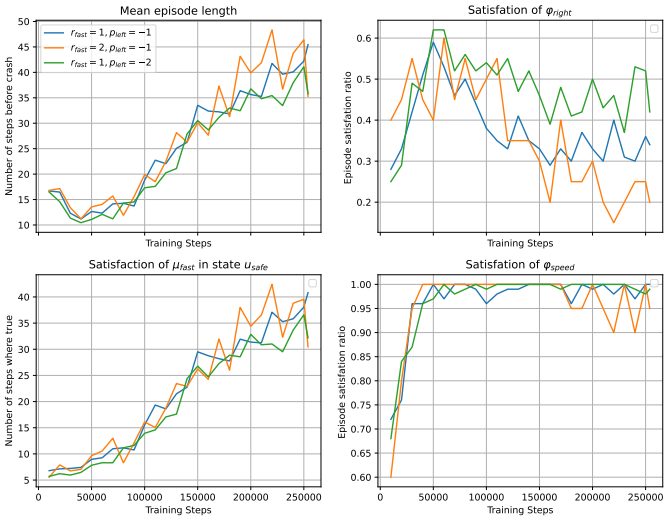


Fig. 10: Training results for different values of rewards and penalties. Every 1e5 steps, the agent is tested for 100 episodes and all formulas are evaluated to give a complete picture of the current performance. All values lead to improving performance, lower values of p_{left} increase the rate of satisfaction of φ_{right}, which is consistent with the formulas and RM.

critical experiences to reflect them in intermediate rewards. This however should be done using a search procedure with good coverage over the behavior space. Another natural direction is STL shielding for safe-RL. Our combination of RM and STL provides a flexible and practical framework to restrict certain critical actions under specific conditions. For instance, in the highway-env case studies, agents should be strictly prohibited from accelerating when μ_{danger} is true. Finally, we aim to extend our framework to recent variable-time-step RL approaches, where actions are executed only when necessary. In this context, the dense-time semantics of STL offers a distinct advantage over other discrete time temporal logics, as they naturally accommodate asynchronous decision-making and continuous-time reasoning.

VI. Acknowledgments

This work is partially supported by the joint French-Japanese ANR-JST project CyPhAI, the Auvergne-Rhône-Alpes Region Project DetAI, and by a French state grant managed by the National Research Agency as part of France 2030, with the reference ANR-23-IACL-0006.

References

[1] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta, "Q-Learning for robust satisfaction of Signal Temporal Logic specifications," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, 2016, pp. 6565–6570.

[2] X. Li, C. I. Vasile, and C. Belta, "Reinforcement learning with temporal logic rewards," *CoRR*, vol. abs/1612.03471, 2016. [Online]. Available: http://arxiv.org/abs/1612.03471

[3] R. Brafman, G. De Giacomo, and F. Patrizi, "LTLf/LDLf Non-Markovian Rewards," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018.

[4] M. Hasanbeig, Y. Kantaros, A. Abate, D. Kroening, G. J. Pappas, and I. Lee, "Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees," 2019. [Online]. Available: https://arxiv.org/abs/1909.05304

[5] G. De Giacomo, L. Iocchi, M. Favorito, and F. Patrizi, "Foundations for Restraining Bolts: Reinforcement Learning with LTLf/LDLf Restraining Specifications," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, 05 2021, pp. 128–136.

[6] E. M. Hahn, M. Perez, S. Schewe, F. Somenzi, A. Trivedi, and D. Wojtczak, "Omega-Regular Objectives in Model-Free Reinforcement Learning," in *Tools and Algorithms for the Construction and Analysis of Systems*, T. Vojnar and L. Zhang, Eds. Cham: Springer International Publishing, 2019, pp. 395–412.

[7] M. Hasanbeig, D. Kroening, and A. Abate, "Deep Reinforcement Learning with Temporal Logics," in *Proceedings of 18th International Conference Formal Modeling and Analysis of Timed Systems FORMATS*, vol. 29, 2020, pp. 1–22.

[8] Z. Xu and U. Topcu, "Transfer of temporal logic formulas in reinforcement learning," *CoRR*, vol. abs/1909.04256, 2019. [Online]. Available: http://arxiv.org/abs/1909.04256

[9] P. Kapoor, A. Balakrishnan, and J. V. Deshmukh, "Model-based Reinforcement Learning from Signal Temporal Logic Specifications," *CoRR*, vol. abs/2011.04950, 2020.

[10] Y.-L. Kuo, B. Katz, and A. Barbu, "Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of LTL formulas," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 5604–5610.

[11] Y. Jiang, S. Bharadwaj, B. Wu, R. Shah, U. Topcu, and P. Stone, "Temporal-logic-based reward shaping for continuing learning tasks," *CoRR*, vol. abs/2007.01498, 2020. [Online]. Available: https://arxiv.org/abs/2007.01498

[12] P. Vaezipoor, A. C. Li, R. T. Icarte, and S. A. McIlraith, "LTL2Action: Generalizing LTL Instructions for Multi-Task RL," in *International Conference on Machine Learning*, 2021.

[13] K. Jothimurugan, R. Alur, and O. Bastani, "A composable specification language for reinforcement learning tasks," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Cham: Springer Nature Switzerland, 2019, pp. 13 041–13 051.

[14] R. Alur, S. Bansal, O. Bastani, and K. Jothimurugan, "A framework for transforming specifications in reinforcement learning," *CoRR*, vol. abs/2111.00272, 2021. [Online]. Available: https://arxiv.org/abs/2111.00272

[15] R. T. Icarte, T. Klassen, R. Valenzano, and S. McIlraith, "Using reward machines for high-level task specification and decomposition in reinforcement learning," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 2107–2116. [Online]. Available: https://proceedings.mlr.press/v80/icarte18a.html

[16] R. T. Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith, "Reward machines: Exploiting reward function structure in reinforcement learning," *CoRR*, vol. abs/2010.03950, 2020. [Online]. Available: https://arxiv.org/abs/2010.03950

[17] A. Camacho, R. Toro Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith, "LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 2019, pp. 6065–6073.

[18] C. Neary, Z. Xu, B. Wu, and U. Topcu, "Reward machines for cooperative multi-agent reinforcement learning," *CoRR*, vol. abs/2007.01962, 2020. [Online]. Available: https://arxiv.org/abs/2007.01962

[19] D. Muniraj, K. G. Vamvoudakis, and M. 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)*, 2018, pp. 4141–4146.

[20] J. Ikemoto and T. Ushio, "Deep Reinforcement Learning Based Networked Control with Network Delays for Signal Temporal Logic Specifications," in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2022, pp. 1–8.

[21] S. Bansal, "Specification-Guided Reinforcement Learning," in *Static Analysis*, G. Singh and C. Urban, Eds. Cham: Springer Nature

Switzerland, 2022, pp. 3–9.
[22] R. T. Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith, "Reward Machines: Exploiting Reward Function Structure in Reinforcement Learning," *CoRR*, vol. abs/2010.03950, 2020.
[23] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, Y. Lakhnech and S. Yovine, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 152–166.
[24] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *Formal Modeling and Analysis of Timed Systems*, K. Chatterjee and T. A. Henzinger, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 92–106.
[25] J. V. Deshmukh, A. Donzé, S. Ghosh, X. Jin, G. Juniwal, and S. A. Seshia, "Robust online monitoring of signal temporal logic," *Formal Methods in System Design*, vol. 51, no. 1, pp. 5–30, 2017.
[26] N. Hamilton, P. K. Robinette, and T. T. Johnson, "Training agents to satisfy timed and untimed signal temporal logic specifications with reinforcement learning," in *20th International Conference on Software Engineering and Formal Methods (SEFM)*, Sep. 2022.
[27] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan, and O. G. Younis, "Gymnasium: A standard interface for reinforcement learning environments," 2024. [Online]. Available: https://arxiv.org/abs/2407.17032
[28] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html
[29] M. Chevalier-Boisvert, B. Dai, M. Towers, R. de Lazcano, L. Willems, S. Lahlou, S. Pal, P. S. Castro, and J. Terry, "Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks," *CoRR*, vol. abs/2306.13831, 2023.
[30] The Farama Foundation - Cart Pole, "Cart Pole," 2023, 2024-10-08.

APPENDIX

TABLE III: PPO Training Hyperparameters

Hyperparameter	**Value**
Total Timesteps	5×10^5
Learning Rate (α)	3×10^{-4}
Rollout Steps (n_{steps})	2048
Batch Size	64
Optimization Epochs (n_{epochs})	10
Discount Factor (γ)	0.99
GAE Lambda (λ)	0.95
Clip Range (ε)	0.2
Network Architecture	[256, 256]