Temporal Logic Monitoring Rewards via Transducers

Giuseppe De Giacomo, Marco Favorito, Luca Iocchi, Fabio Patrizi, Alessandro Ronca
DIAG, Università di Roma “La Sapienza”, Italy
{lastname}@diag.uniroma1.it

Abstract
In Markov Decision Processes (MDPs), rewards are assigned according to a function of the last state and action. This is often limiting, when the considered domain is not naturally Markovian, but becomes so after careful engineering of an extended state space. The extended states record information from the past that is sufficient to assign rewards by looking just at the last state and action. Non-Markovian Reward Decision Processes (NRMDPs) extend MDPs by allowing for non-Markovian rewards, which depend on the history of states and actions. Non-Markovian rewards can be specified in temporal logics on finite traces such as \( \text{LTL}_f/\text{LDL}_f \), with the great advantage of a higher abstraction and succinctness; they can then be automatically compiled into an MDP with an extended state space. We contribute to the techniques to handle temporal rewards and to the solutions to engineer them. We first present an approach to compiling temporal rewards which merges the formula automata into a single transducer, sometimes saving up to an exponential number of states. We then define monitoring rewards, which add a further level of abstraction to temporal rewards by adopting the four-valued conditions of runtime monitoring; we argue that our compilation technique allows for an efficient handling of monitoring rewards. Finally, we discuss applications to reinforcement learning.

1 Introduction
In a Markov Decision Process (MDP) (Puterman 1994) the transition probability function and the reward function are Markovian, i.e., they depend only on the last state and action. However, this limitation does not allow for rewarding behaviours that extend overtime; alternatively, it requires to engineer an extended state space where states record enough information from the past. To overcome such limitations, non-Markovian Reward Decision Processed (NRMDP) have been proposed (Bacchus, Boutilier, and Grove 1996; Thibaux et al. 2006). In particular, the idea is to encode non-Markovian rewards into an MDP by extending the state space, with minimality guarantees of the resulting MDP.

The same idea, with some variations, has been investigated in more recent works. In (Icarte et al. 2018; Camacho et al. 2019), the authors introduce the concept of reward machine, an automata-based formalism to encode non-Markovian rewards, that has been successfully applied in a multi-task setting. In (Quint et al. 2019), formal languages are used to specify soft and hard constraints on actions, by enforcing constraints on the action space, called action shaping. In (Alshiekh et al. 2018), an approach based on temporal logic has been used to monitor the actions of an agent and to prevent the violation of critical safety specifications. In (Brafman, De Giacomo, and Patrizi 2018; De Giacomo et al. 2019), rewards are specified in the temporal logics \( \text{LTL}_f/\text{LDL}_f \) (De Giacomo and Vardi 2013; De Giacomo and Vardi 2015; De Giacomo and Vardi 2016). Here the construction of the extended MDP is based on the correspondence between such logics and finite-state automata (Rabin and Scott 1959). Specifically, the extended MDP is obtained as the synchronous product of a formula’s automata with the automata underlying the NMRDP. All of these techniques are examples of how much Knowledge Representation can be of great help for reward specification.

A crucial property of such techniques is the overhead required to handle the non-Markovianity. Such overhead is introduced in the original state space to generate the extended MDP over which the learning is performed. It is desirable that the overhead is the minimum possible since it affects the effectiveness of learning algorithms (e.g. the exploration phase in Reinforcement Learning (Sutton and Barto 2018)).

In this paper, we want to extend the approach in (Brafman, De Giacomo, and Patrizi 2018) while keeping such overhead to the minimum. To do so, we merge automata of the various formulas used for the rewards into a single transducer from traces to rewards (i.e. outputs a reward for every prefix of the trace), which encodes all the temporal specifications in a single finite-state machine. This gives us further opportunities of minimizations if we do not care from the satisfaction of which formula a given reward is obtained. Indeed, we show that by giving up this information, the transducer can be exponentially (in fact factorially) smaller than the minimal automaton in (Brafman, De Giacomo, and Patrizi 2018), and never worse in general. Then, inspired by the literature on monitoring (Bauer, Leucker, and Schallhart 2010; Ly et al. 2013; De Giacomo et al. 2014), we devise a way of specifying rewards using \( \text{LTL}_f/\text{LDL}_f \) which associates reward not to simply the satisfaction of the formula, but to the four classical monitoring conditions: the formula is temporarily true, temporarily false, permanently true, and permanently false. We illustrate the convenience of this kind of
We discuss the use of such kind of LTL/LDLf-based reward specifications in reinforcement learning of non-Markovian specifications.

2 Background

MDPs and RL. A Markov Decision Process (MDP) $M = \langle S, A, Tr, R \rangle$ contains a set $S$ of states, a set $A$ of actions, a transition function $Tr: S \times A \rightarrow Prob(S)$ that returns for every state $s$ and action $a$ a distribution over the next state, and a reward function $R: S \times A \rightarrow \mathbb{R}$ that specifies the reward (a real value) received by the agent when transitioning from state $s$ to state $s'$ by applying action $a$. We see states $S$ as truth assignments to a set $P$ of propositional atoms. A solution to an MDP is a function, called a policy, assigning an action to each state, possibly with a dependency on past states and actions. The value of a policy $\rho$ at state $s$, denoted $v^\rho(s)$, is the expected sum of (possibly discounted by a factor $\gamma$, with $0 \leq \gamma \leq 1$) rewards when starting at state $s$ and selecting actions based on $\rho$. Typically, the MDP is assumed to start in an initial state $s_0$, so policy optimality is evaluated w.r.t. $v^\rho(s_0)$. Every MDP has an optimal policy $\rho^*$. In discounted cumulative settings, there exists an optimal policy that is Markovian $\rho: S \rightarrow A$, i.e., $\rho$ depends only on the current state, and deterministic (Puterman 1994).

Reinforcement Learning (RL) is the task of learning a possibly optimal policy, from an initial state $s_0$, on an MDP where only $S$ and $A$ are known, while $Tr$ and $R$ are not—see, e.g., (Sutton and Barto 2018).

Automata. A (finite-state) automaton is a computational model with limited capabilities. It can read input strings in a given alphabet, it keeps track of its current state among finitely many, and it can produce output strings. An automaton whose output response is limited to a simple ‘yes’ or ‘no’ is called an acceptor. A more general automaton, capable of producing strings of symbols as output, is called a transducer.

The most basic kind of automata are deterministic finite automata (DFA) (Rabin and Scott 1959). A DFA is a 5-tuple $A = \langle Q, \Sigma, q_0, F, \delta \rangle$ where $Q$ is the (non-empty) finite set of states, $\Sigma$ is the set of input symbols, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states, and $\delta: Q \times \Sigma \rightarrow Q$ is the transition function. The extended transition function $\delta^*: A = \delta^*(q, \varepsilon) = q$ and $\delta^*(q, aw) = \delta(\delta^*(q, a), w)$, $a$. Automaton $A$ accepts a word $w$ if $\delta^*(q_0, w) \in F$. The language of $A$, written $L(A)$, is the set of words that $A$ accepts.

Two fundamental kinds of transducers are Moore machines and Mealy machines. A Moore machine (Moore 1956) is a tuple $M_o = \langle Q, \Sigma, \Gamma, q_0, \delta, \theta \rangle$ where $Q$ is the set of states, $\Sigma$ is the set of input symbols, $\Gamma$ is the set of the output symbols, $q_0$ is the initial state, $\delta: Q \times \Sigma \rightarrow Q$ is the transition function, and $\theta: Q \rightarrow \Gamma$ is the output function that maps states to output symbols. The output of $M_o$ on word $a_1 \ldots a_n$ is $\theta(q_0) \theta(\delta^*(q_0, a_1)) \ldots \theta(\delta^*(q_0, a_1, \ldots, a_n))$.

A Mealy machine $M_e$ (Mealy 1955) is like a Moore machine except that its output function $\theta: Q \times \Sigma \rightarrow \Gamma$ maps transitions to output symbols, instead of states. Hence the output of $M_e$ on word $a_1 \ldots a_n$ is $\theta(q_0, a_1) \theta(\delta^*(q_0, a_1, a_2)) \ldots \theta(\delta^*(q_0, a_1, \ldots, a_n-1), a_n)$. Note that, for a Moore machine and a Mealy machine performing the same number of transitions, the output of the Mealy has one symbol less. A Moore/Mealy machine $M$ corresponds to the transduction function $F_M: \Sigma^* \rightarrow \Gamma^*$ that maps its input to its output. That is, such machines translate words on the input alphabet $\Sigma$ to words on the output alphabet $\Gamma$. It can be shown that Moore machines and Mealy machines have the same expressivity, that is, for a Moore machine there exist an equivalent Mealy machine, and vice versa (Linz 2006).

An important property that we will use in the next sections is that both DFAs and Moore/Mealy machines can be minimised, and the resulting minimal automata are unique (modulo state renaming) for the language they recognise or the transduction function they represent, respectively.

$LTL/\text{LDLf}$. The logic $LTL$ is the classical linear time logic (Pnueli 1977) interpreted over finite traces, formed by a finite (instead of infinite, as in $LTL$) sequence of propositional interpretations (De Giacomo and Vardi 2013). The underlying propositional alphabet we consider here is the one given by the world features Given a set $P$ of propositional atoms, $LTL_f$ formulas $\varphi$ are defined as follows:

$$\varphi ::= \phi \mid \neg \varphi \mid \varphi_1 \land \varphi_2 \mid \circ \varphi \mid \varphi_1 U \varphi_2$$

where $\phi$ is a propositional formula over $P$, $\circ$ is the next operator and $U$ is the until operator. We use the standard abbreviations: $\varphi_1 \lor \varphi_2 = \neg(\neg \varphi_1 \land \neg \varphi_2)$; eventually as $\Diamond \varphi = \text{true} U \varphi$; always as $\Box \varphi = \neg \Diamond \neg \varphi$; week next $\bullet \varphi = \neg \circ \neg \varphi$ (note that on finite traces $\neg \circ \varphi \neq \neg \Diamond \varphi$); and last $\text{false}$ denoting the end of the trace. $LTL_f$ is as expressive as first-order logic over finite traces and star-free regular expressions.

$LDLf$ is a proper extension of $LTL_f$, which is as expressive as monadic second-order logic over finite traces and (unrestricted) regular expressions (De Giacomo and Vardi 2013). An $LDLf$ formula $\varphi$ is built as follows:

$$\varphi ::= \text{tt} \mid \neg \varphi \mid \varphi_1 \land \varphi_2 \mid \langle \varrho \rangle \varphi \mid \varphi_1 \varrho \varphi_2 \mid \varrho^*$$

where: $\text{tt}$ stands for logical true; $\varphi$ is a propositional formula over $P$; $\circ$ denotes path expressions, i.e., $RE$ over propositional formulas $\varphi$ with the addition of the test construct $\varrho$ typical of Propositional Dynamic Logic ($PDL$). We use abbreviations $\langle \varrho \rangle \varphi = \neg(\neg \varphi) \neg \varphi$ as in $PDL$. Intuitively, $\langle \varrho \rangle \varphi$ states that, from the current step in the trace, there exists an execution satisfying the $RE$ $\varrho$ such that its last step satisfies $\varphi$, while $[\varrho] \varphi$ states that, from the current step, all executions satisfying the $RE$ $\varrho$ are such that their last step satisfies $\varphi$. Tests are used to insert into the execution path checks for satisfaction of additional $LDLf$ formulas.

A remarkable property of $LTL_f/\text{LDLf}$ is that, for each formula $\varphi$, we can construct a DFA $A_\varphi$ that tracks satisfaction
of $\varphi$: $A_{\varphi}$ accepts a finite trace $\pi$ if $\pi$ satisfies $\varphi$. Our results crucially rely on the existence of such an automaton. However, this is not unique to LTL/l/\text{LDL}_f. An analogous transformation to automata applies to several other formalisms for representing temporal specifications over finite traces, including Past LTL, co-safe LTL, etc. (Baccus, Boutilier, and Grove 1996; Thiebaux et al. 2006; Slaney 2005; Gretton 2007; Gretton et al. 2014; Lacerda, Parker, and Hawes 2015).

NMRDPs. A non-Markovian reward decision process (NMRDP) (Baccus, Boutilier, and Grove 1996) is a tuple $(S, A, Tr, R)$, where $S$, $A$ and $Tr$ are as in an MDP (with each in $S$ being an assignment for propositions $P$), but the reward $R$ is a real-valued function over finite state-action sequences (referred to as traces), i.e., $R : (S \times A)^* \rightarrow \mathbb{R}$. Given a (possibly infinite) trace $\pi = (s_0, a_1, \ldots, s_{n-1}, a_n)$, the value of $\pi$ is: $v(\pi) = \sum_{i=1}^{n}[\pi] R(\langle (\pi(1), (\pi(2), \ldots, (\pi(i))\rangle)$, where $\pi(i)$ denotes the pair $(s_i, a_i)$. In NMRDPs, policies are also non-Markovian $\rho : S^* \rightarrow A$. Since every policy induces a distribution over the set of possible infinite traces, we can define the value of a policy $\rho$, given an initial state $s$, as: $v^\rho(s) = E_{\pi \sim M, \rho, s} v(\pi)$. That is, $v^\rho(s)$ is the expected value of infinite traces, where the distribution over traces is defined by the initial state $s$, the transition function $Tr$, and the policy $\rho$.

Specifying a non-Markovian reward function explicitly is cumbersome and unintuitive, even if only a finite number of traces are to be rewarded. LTL/l/\text{LDL}_f provides an intuitive and convenient language for non-Markovian rewards (Camacho et al. 2017; Brafman, De Giacomo, and Patrizi 2018). Following (Brafman, De Giacomo, and Patrizi 2018) we can specify $\rho$ using a set of pairs $\{(\varphi_i, r_i)\}_{i=1}^m$, where each $\varphi_i$ is an LTL/l/\text{LDL}_f formula over the propositions $P$ that selects the traces to reward, and $r_i$ the reward assigned to those traces. When the current (partial) trace is $\pi = (s_0, a_1, \ldots, s_{n-1}, a_n)$, the agent receives at $s_n$ each reward $r_i$ whose formula $\varphi_i$ is satisfied by $\pi$.

From NMRDPs to MDPs. In (Brafman, De Giacomo, and Patrizi 2018) it is shown that for any NMRDP $M = (S, A, Tr, \{(\varphi_i, r_i)\}_{i=1}^m)$, with $\varphi_i$ being LTL/l/\text{LDL}_f formulas, there exists an MDP $M' = (S', A, Tr', R')$ that is equivalent to $M$ in the sense that the states of $M$ can be (injectively) mapped into those of $M'$ in such a way that corresponding (under the mapping) states yield same transition probabilities, and corresponding traces have same rewards (Baccus, Boutilier, and Grove 1996). Denoting with $A_{\varphi_i} = (Q_i, 2^{P \cup A}, q_0, \delta_i, F_i)$ (notice that $S \subseteq 2^P$ and $\delta_i$ is total) the DFA associated with $\varphi_i$, the equivalent MDP $M'$ is as follows:

- $S' = Q_1 \times \cdots \times Q_m \times S$;
- $Tr' : S' \times A \times S' \rightarrow [0, 1]$ is defined as:

$$Tr'(q_1, \ldots, q_m, s, a, q'_1, \ldots, q'_m, s') = \begin{cases} Tr(s, a, s') & \text{if } \forall i : \delta_i(q_i, (s, a)) = q'_i \\ 0 & \text{otherwise} \end{cases}$$

- $R' : S' \times A \rightarrow \mathbb{R}$ is defined as:

$$R'(q_1, \ldots, q_m, s, a) = \sum_{i : q_i \in F_i} r_i$$

Observe that the state space of $M'$ is the product of the state spaces of $M$ and $A_{\varphi_i}$, and that the reward $R'$ is Markovian. In other words, the (stateful) structure of the LTL/l/\text{LDL}_f formulas $\varphi_i$ used in the (non-Markovian) reward of $M$ is compiled into the states of $M'$.

Theorem 1 ((Brafman, De Giacomo, and Patrizi 2018)). The NMRDP $M = (S, A, Tr, \{(\varphi_i, r_i)\}_{i=1}^m)$ is equivalent to the MDP $M' = (S', A, Tr', R')$ defined above.

Actually this theorem can be refined, into a stronger lemma. A policy $\rho$ for an NMRDP $M$ and a policy $\rho'$ for an equivalent MDP $M'$ are equivalent if they guarantee the same rewards. Assume $M'$ is constructed as above and let $\rho'$ be a policy for $M'$. Consider a trace $\pi = (s_0, a_1, s_1, \ldots, s_{n-1}, a_n)$ of $M$ and assume it leads to state $s_n$. Further, let $q_i$ be the state of $A_{\varphi_i}$ on input $\pi$. We define the (non-Markovian) policy $\hat{\rho}$ equivalent to $\rho'$ as $\hat{\rho}(\pi) = \rho'(q_1, \ldots, q_m, s_n)$. Similarly, given a policy $\rho$ for $M$, by just tracking the state of the DFAs $A_{\varphi_i}$, it is immediate to define the equivalent policy $\rho'$ for $M'$. Hence we have:

Lemma 1 ((Brafman, De Giacomo, and Patrizi 2018)). Given an NMRDP $M$ and an equivalent MDP $M'$, every policy $\rho$ for $M$ has an equivalent policy $\rho'$ for $M$ and viceversa.\footnote{A variant of this lemma, talking about optimal policy only as originally presented in (Baccus, Boutilier, and Grove 1996).}

Moreover, as observed by (De Giacomo et al. 2019), it is possible to do RL over the $M'$ equivalent to $M$. Being $M'$ an MDP, this can be done by off-the-shelf RL algorithms (e.g. Q-learning and SARSA). Of course, neither $M$ nor $M'$ are (completely) known to the learning agent, and the transformation is never done explicitly. Rather, during the learning process, the agent assumes that the underlying model has the form of $M'$ instead of that of $M$.

Theorem 2 ((De Giacomo et al. 2019)). RL for LTL/l/\text{LDL}_f rewards over an NMRDP $M = (S, A, Tr, \{(\varphi_i, r_i)\}_{i=1}^m)$, with $Tr$ and $\{(\varphi_i, r_i)\}_{i=1}^m$ hidden to the learning agent can be reduced to RL over the MDP $M' = (S', A, Tr', R')$ defined above, with $Tr'$ and $R'$ hidden to the learning agent.

Runtime Monitoring. We will borrow the four-valued semantics of runtime monitoring on finite traces (Bauer, Leucker, and Schallhart 2010; Lý et al. 2013; De Giacomo et al. 2014). Given an LTL/l/\text{LDL}_f formula $\varphi$ and a trace $\pi$, we say that:

- $\varphi$ is temporarily true in $\pi$ if $\pi$ satisfies $\varphi$ and there is a continuation of $\pi$ that does not satisfy $\varphi$, and we write $\pi = [[\varphi = \text{temp\_true}]]$;
- $\varphi$ is temporarily false in $\pi$ if $\pi$ does not satisfy $\varphi$ and there is a continuation of $\pi$ that satisfies $\varphi$, and we write $\pi = [[\varphi = \text{temp\_false}]]$.
• $\varphi$ is permanently true in $\pi$ if $\pi$ and all its continuations satisfy $\varphi$, and we write $\pi \models [\varphi = \text{perm_true}]$.
• $\varphi$ is permanently false in $\pi$ if $\pi$ and all its continuations do not satisfy $\varphi$, and we write $\pi \models [\varphi = \text{perm_false}]$.

3 Reward Transducers

In this section, we give the definition of reward transducers and define some operations over them, that will be used in later sections.

In our context, a reward transducer maps MDP traces of the form $\pi = \langle (s_0, a_1), (s_1, a_2), \ldots, (s_n, a_n) \rangle$ to sequences of rewards $r_1, r_2, \ldots, r_n$ (i.e. it outputs a reward for every prefix of the trace).

Definition 1. For states $S$ and actions $A$, a reward transducer is a transducer with input alphabet $S \times A$ and output alphabet $R \subseteq \mathbb{R}$. A Moore (or Mealy) reward machine is a reward transducer that is a Moore (Mealy) machine.

Thus, we can define any non-Markovian reward function $R$ as a transduction function and we can implement it as a transducer.

With that spirit, we observe that a temporal specification $(\varphi, r)$ can be transformed into an equivalent reward transducer. Indeed, we can transform the associated DFA $A_\varphi$ into the Moore machine $M_\varphi = (Q, 2^P \times A, \{0, r\}, q_0, \delta, \theta_0)$ where $Q$, $q_0$ and $\delta$ are defined as in $A_\varphi$, and $\theta_0(q) = 0$ if $q \notin F$. $\theta_1(q) = r$ otherwise. That is, the Moore machine outputs 0 for every prefix that does not satisfy $\varphi$ and outputs $r$ for every prefix that satisfies it. Analogously we can define a Mealy machine $M_\varphi^2 = (Q, 2^P \times A, \{0, r\}, q_0, \delta, \theta_e)$ where everything is defined like in the Moore machine but $\theta_e(q, (s, a)) = r$ if $\delta(q, (s, a)) \in F$, and 0 otherwise.

Sum of Reward Transducers. We now define the sum of two Moore reward machines $M_1 = (Q_1, \Sigma, R_1, q_{01}, \delta_1, \theta_1)$ and $M_2 = (Q_2, \Sigma, R_2, q_{02}, \delta_2, \theta_2)$, which is a Moore reward machine outputting the sum of the rewards of the two initial machines. Formally, the sum machine $M'_o = M_1 + M_2 = (Q, \Sigma, R, q_0, \delta, \theta_o)$ where:

• $Q = Q_1 \times Q_2$;
• $R = \{r_1 + r_2 \mid r_1 \in R_1, r_2 \in R_2\}$;
• $q_0 = (q_{01}, q_{02})$;
• $\delta = \{(q_1, q_2) \xrightarrow{\sigma} (q_1', q_2') \mid \forall i \in \{1, 2\}. q_i \xrightarrow{\sigma} q_i' \in \delta_i\}$;
• $\theta_o = \{(q_1, q_2) \mapsto r_1 + r_2 \mid \forall i \in \{1, 2\}. \theta_i(q_i) = r_i\}$.

It is a standard cross-product construction, where the output of each new state is the sum of the outputs of the corresponding old states. As a result of the construction, the transduction function implemented by $M'_o + M_2^2$ is the sum of the functions of $M_1$ and $M_2^2$, i.e., $F_{M'_o + M_2^2} = F_{M'_o} + F_{M_2^2}$.

The sum of two Mealy reward machines is defined very similarly to the sum of two Moore reward machines. The only thing that changes is the output function is built:

$\theta_e = \{(q_1, q_2, \sigma) \mapsto r_1 + r_2 \mid \forall i \in \{1, 2\}. \theta_i(q_i, \sigma) = r_i\}$.

Direct Sum of Reward Transducers. If the two machines are basically the same machine except for the output function, then we can build a sum machine simply by taking the sum of their output function. In this case we call the two machines shape-equivalent—a notion inspired by the shape-equivalence for DFAs in (De Giacomo and Rubin 2018). Specifically, $M_1^o$ and $M_2^o$ are shape-equivalent if differ only in their output, or in other words have the same states, input alphabet, initial state, and transition function. For such machines, we can then define the direct sum machine $M_1^o + M_2^o = (Q, \Sigma, R, q_0, \delta, \theta)$ where $Q$, $q_0$, and $\delta$ are the common states, initial state, and transition function, respectively, $R$ is defined as for $M_1^o + M_2^o$, and $\theta = \theta_1 + \theta_2$. It is again the case that $F_{M_1^o + M_2^o} = F_{M_1^o} + F_{M_2^o}$. Whenever in the following we take the sum of two machines, we can instead take their direct sum if we know that they are shape-equivalent. The same definition applies to the Mealy reward machines, except that the transition function depends on a state-symbol pair, rather than just a state.

In Figure 1, we show the Moore reward machines for the temporal specifications $\langle \Diamond 0, +1 \rangle$ and $\langle \Box b, +2 \rangle$ and their sum. In Figure 2, we show the equivalent Mealy reward machines.

4 Extending MDPs via Reward Transducers

Rewarding complex behaviours is a challenging task, and temporal logic provides the right level of abstraction to address the problem (Littman 2015; Littman et al. 2017). This is the philosophy behind NMRDPs with LTL/LDLf rewards (Brafman, De Giacomo, and Patrizi 2018). NMRDPs can be reduced to MDPs, and hence solved using off-the-shelf algorithms for MDPs. This comes, however, at the cost of an extension of the state space, which is required to keep track of the state of partial satisfaction of the temporal rewards. Such an extension introduces an overhead which is necessary to deal with non-Markovianity, but it is a computational cost for the algorithm that has to solve the resulting MDP. It is then important to keep such an overhead to a minimum.
Figure 2: Here we show the same specifications depicted in Figure 1, but implemented as Mealy reward machines. In Figure (a), the Mealy reward machine $M^1_o$ for the temporal specification $\langle a, +1 \rangle$. In Figure (b), Moore reward machine $M^2_o$ for the temporal specification $\langle \exists b, +2 \rangle$. In Figure (c), the Mealy machine of the sum of $M^1_o$ and $M^2_o$.

Definition 2. Given an NMRDP $M$ with state space $S$ and an equivalent MDP $M'$ with state space $S'$, the state overhead of $M'$ on $M$ is $|S'| - |S|$

In this section, we propose a novel construction for the extended MDP, that achieves a significantly smaller state overhead by using reward transducers introduced in the previous section, instead of DFAs, to assign rewards. In particular, we can define an MDP that plays the same role as the one described in (Brafman, De Giacomo, and Patrizi 2018), with the exception that does not keep track of which formula the reward comes from. We use a Moore reward machine, which is the sum of the Moore machines for the single rewards—rather than the cross product of the DFAs for the reward formulas.

Consider an NMRDP $M = \langle S, A, Tr, \{ (\varphi_i, r_i) \}_{i=1}^m \rangle$, and let $M^\varphi_i$ be the Moore reward machine for $\varphi_i$. We define $M_o = \langle Q_o, 2^P \times A, R_o, q_0, \delta_o, \theta_o \rangle$ as the sum of all the other Moore reward machines, i.e., $M_o = M^\varphi_1 \times \cdots \times M^\varphi_m$. We derive the new MDP $M_o = \langle S_o, A, Tr_o, R_o \rangle$ as follows:

- $S_o = S \times Q$;
- $Tr_o : S_o \times A \times S_o \to [0, 1]$ is defined as:
  \[ Tr_o((q,s), a, (q', s')) = \begin{cases} Tr(s, a, s') & \text{if } \delta_o(q, (s, a)) = q' \\ 0 & \text{otherwise} \end{cases} \]
- $R_o : S_o \times A \to \mathbb{R}$ is defined as:
  \[ R_o((q,s), a) = \theta_o(q') \]

Note that $M_o$ is equivalent to $M'$ as for Theorem 1, and hence to $M$. We formalize this observation in the following theorem.

Theorem 3. The NMRDP $M$ and the MDP $M_o$ are equivalent.

Proof. We need to prove that $M_o$ is equivalent to $M'$, since the equivalence between $M$ and $M'$ is a consequence of Theorem 1. Notice that, by construction, $S_o$ is isomorphic to $S'$, and so is $Tr_o$ to $Tr'$, due to the definition of $\delta$. Finally, notice that $\varphi_l(q) = \sum_i \theta_l(q_i)$, where $\theta_l(q_i) = r_i$ when $q_i \in F_l$, so $R_o$ is simply a compact representation of $R'$.

Even assuming that the machines $M^\varphi_i$ are minimal, their sum machine $M_o$ may not be. So $M_o$ may need to undergo a minimisation step if we desire to minimise the state space of the resulting MDP $M_o$, to minimise the state overhead of $M_o$ on $M$.

Now that we have defined the extended MDP construction based on Moore machines we show that such a construction can significantly reduce the state overhead. In fact, it can achieve an exponential improvement (in fact, factorial), as argued in the following theorem.

Theorem 4. For every $n \geq 1$, there is an NMRDP $M$ such that (i) the equivalent MDP $M$ (as in Theorem 1) has state overhead $\Omega(n!)$ on $M$, and (ii) the equivalent MDP $M_o$ (as introduced this section) has state overhead $O(n)$ on $M$.

Proof. Consider a set of propositions $\mathcal{P} = \{ p_1, \ldots, p_n \}$ and an NMRDP $M = \langle 2^\mathcal{P}, \{ \text{ins}, \text{del} \} \times A, Tr, \{ (\varphi_i, 1) \}_{i=1}^n \rangle$ where each $\varphi_i$ is of the form:

$\Omega(\neg p_1 \land \cdots \land \neg p_i-1 \land p_i \land \neg p_i+1 \land \cdots \land \neg p_n)$

and $Tr$ consists of transitions

$(s, (\text{ins}, p)) \mapsto (s \cup \{ p \})$ and $(s, (\text{del}, p)) \mapsto (s \setminus \{ p \})$.

for each $p \in \mathcal{P}$ and each $s \in 2^\mathcal{P}$. Intuitively, we can insert and delete propositions to/from states, and at the $i$-th step we get rewarded if $p_i$ is true and the other propositions are false. The minimum DFA for $\varphi_i$ has $\Omega(i)$ states. As a result, the MDP $M'$ has state overhead $\Omega(n!)$. Now we argue that the state overhead of $M_o$ is $O(n)$. A Moore reward machine $M^\varphi_{\varphi_i}$ for $\varphi_i$ has states $s_0, \ldots, s_i$, and a transition from $s_j$ to $s_{j+1}$ for each $j \leq i - 1$ and each input symbol, and it outputs 0 in all transitions but the last one, where it outputs 1 if it reads $\{ p_i \}$. Reward machines $M^\varphi_{\varphi_i}$ can be summed into one machine having states $s_0, \ldots, s_n$, and transitions from $s_j$ to $s_{j+1}$ for each $j \leq n - 1$ and each input symbol, with the output at the $i$-th transition being 1 for input $\{ p_i \}$, and 0 otherwise.

Moreover the approach based on transducers never does worse than the one based on DFAs.

Theorem 5. For every NMRDP $M$, the equivalent MDP $M_o$ (as introduced in this section) has state overhead smaller than or equal to the state overhead of the MDP $M'$ (as in Theorem 1).

Proof. It suffices to notice that in both cases we can build an extended state based on the cross product of the states of automata for the reward formulas. 

Considering that our goal is to keep the state overhead to a minimum, we next focus on Mealy reward machines. We first observe that Mealy reward machines are similar to the reward machines as defined in (Camacho et al. 2019). However, whilst in their work they are more focused on the problem of using the machine-based approach, we highlight
the advantages of the reward specification at logical level. As we will show in the later section, sometimes it is exponentially harder to specify a reward machine rather than its equivalent logical formula.

Mealy machines will allow us to save on states, since they can represent Moore machines using possibly less states and never more. In particular, we define a construction that leverages Mealy reward machines, instead of Moore reward machines. Note that every Moore machine can be transformed into a Mealy machine by composing its output function $\theta_e$ with its transition function $\delta_o$, to get a new output function $\theta_e = \theta_o(\delta_o(s, a))$. Hence, from the MDP $M_o = (S_o, A, Tr_o, R_o)$, we can construct a new MDP $M_e = (S_e, A, Tr_e, R_e)$ where everything is defined as in $M_o$ except $R_e : S_e \times A \rightarrow \mathbb{R}$ that is defined as:

$$R_e((q, s), a) = \theta_e(\delta_o(q, (s, a)))$$

**Theorem 6.** The NMRDP $M$ and the MDP $M_e$ are equivalent.

**Proof.** By construction, $M_e$ is equivalent to $M_o$, and by Theorem 3 and Theorem 1, the thesis follows. $\square$

## 5 Rewards as Temporal Specifications

In this section we argue for the case of the temporal logics LTL$_f$/LDL$_f$ as an appropriate language to specify rewards. In particular, they capture Markovian rewards without loss of efficiency (using transducers) while in general being succinct.

### Capturing Markovian rewards
We start by showing how any MDP $M$ can be represented as an NMRDP $M_{nmr}$ without loss of efficiency in terms of number of states. Specifically, we mean that $M_{nmr}$ has the same states as $M$, and $M_{nmr}$ can be automatically encoded back into an MDP $M'$ which has again the same states of the original MDP $M$.

Consider an MDP $M = (S, A, Tr, R)$. Such an MDP is captured by the following NMRDP:

$$M_{nmr} = (S, A, Tr, \{(\varphi_{(s, a)}, R(s, a))\}_{s \in S, a \in A})$$

where $\varphi_{(s, a)}$ has the form $\Diamond{s \land a \land \text{last}}$—note that $R(s, a)$ is the Markovian reward when the last state and action are $s$ and $a$, respectively. First, we argue that $M_{nmr}$ correctly encodes the given MDP $M$.

**Theorem 7.** The MDP $M$ and the NMRDP $M_{nmr}$ are equivalent.

**Proof.** By construction, the non-Markovian rewards depend only on the last state-action pair, i.e., they are Markovian, although formalized as non-Markovian. Hence, from a non-Markovian policy $\bar{\rho}$, we can build an equivalent Markovian policy $\rho$ by ignoring the history but the last state. Analogously, from $\rho$ we can define a $\bar{\rho}$ such that for all the possible traces $\pi$ that end up in the same state $s$, we have $\bar{\rho}(\pi) = \rho(s)$. $\square$

For further clarity, in Figure 3 is depicted the DFA corresponding to a generic $\varphi_{(s, a)}$.

![Figure 3: Automaton corresponding to $\varphi_{(s, a)} = \Diamond{s \land a \land \text{last}}$](image)

Then, we can convert the NMRDP $M_{nmr}$ into an equivalent MDP $M_e$ using a construction based on a Mealy machine, as discussed in Section 3. The involved Mealy machine is a straightforward state-less encoding of $R$, and it is depicted in Figure 4. Most importantly, $M_e$ has state space $S_e = S \times Q_e = S \times \{q_0\}$ that is isomorphic to the original state space $S$, given the fact that $Q_e$ is a singleton. This shows two points in favour of our transducer-based approach: (i) It is able to fully capture Markovian reward functions, at no cost of additional states; (ii) It is a significant improvement over the construction based on DFAs (Brafman, De Giacomo, and Patrizi 2018) (see Theorem 1 in the background section), since it allows to handle Markovian rewards seamlessly without incurring an exponential blow-up of the state space (which in the DFA-based approach is due to the Cartesian product of the reward automata $A_e$).

### Capturing reward machines
It has been argued (Cama-cho et al. 2019) that reward machines should be the ‘lingua franca’ for non-Markovian rewards. However, LTL$_f$/LDL$_f$ specifications are equally expressive and might be considered more intelligible.

We start by showing that LTL$_f$ reward specifications capture reward machines without loss of efficiency. Specifically, we show that every reward machine can be described by a set of LTL$_f$ specifications, and that the original reward machine can be computed back from those specifications by applying the construction for shape-equivalent machines shown in Section 3. This shows that the higher level of abstraction of temporal specifications comes at no extra cost in terms of number of states of the underlying automata.

Consider a Moore reward machine $M_o = (Q, \Sigma, R, q_0, \delta, \theta_o)$. For each $q \in Q$, let $A_q$ be the DFA $(Q, \Sigma, q_0, \{q\}, \delta)$. We know that there is an LTL$_f$ formula $\varphi_q$, that captures the language $L(A_q)$. Therefore, the Moore reward machine $M_o$ is captured by the temporal specifications $\{(\varphi_q, \theta_o(q))\}_{q \in Q}$. If apply our construction to $\{(\varphi_q, \theta_o(q))\}_{q \in Q}$, using direct sum since the machines for $\varphi_q$ are shape-equivalent, we obtain exactly the initial machine $M_o$. The case of a Mealy reward machine $M_e$ is similar, except that the specifications capturing it are $\{(\varphi_q, \theta_e(q, \sigma))\}_{q \in Q, \sigma \in \Sigma}$.

Furthermore, there are non-Markovian rewards for which LTL$_f$/LDL$_f$ specifications are doubly-exponentially more succinct than reward machines.

**Theorem 8.** There is a family of non-Markovian rewards $R_n$ that admit an LTL$_f$ specification of size $O(n^2)$ and only reward machines of size $\Omega(2^{2n})$.

**Proof.** Consider a set $A$ of at least two actions that an agent can perform. In addition, the agent can perform an action
environment (Moore 1991) reached the position 0.5 ward 1.0.” The car is at a position greater or equal than 0.5, then give

using the If-This-Then-That pattern (IFTTT), namely

transition-based reward function

literature and in the research community. The traditional

LTL

tion

R

Specifying common rewards in logic. It is often the case

that we can represent the same transition-based reward func-

tion $R$ with much less effort, by specifying a non-Markovian

reward fully specified by a much more intelligible $\mathit{LTL}_f$

formula. For example, consider the Mountain Car envi-

ronment (Moore 1991)\textsuperscript{2}, a well-known problem in the RL

literature and in the research community. The traditional

transition-based reward function $R$ is usually implemented

using the If-This-Then-That pattern (IFTTT), namely “if the

car is at a position greater or equal than 0.5, then give re-

ward $1.0$”. By Theorem 7, we know we can always repres-

ent such reward function with temporal specifications. The

reward function of that environment can be represented using

the $\mathit{LTL}_f$ formula $\varphi = \diamond p$, where $p$ means the car has

reached the position 0.5, a proposition opportunely extracted

from the state space. By translating the formula into a DFA

$A_\varphi$ and employing the compilation into the equivalent MDP

as explained in Section 2, we make such non-Markovian re-

ward learnable by classic RL algorithms. It turns out that,

by associating a reward $r_\varphi = 1.0$, the non-Markovian

reward function is exactly the same to the transition-based $R$.

However, there is a clear gap between the two in terms of

intelligibility.

To pursue the analogy with software engineering: the “raw”

$R$ is binary language, the equivalent \{(\varphi(s,a), R(s,a))\}$_{s,a \in A}$

is the decompiled program in a high-level language, say the C++ language, and the

$\mathit{LTL}_f$ formulas are programs written in that language. We

advocate that temporal specifications using proper formal

languages becomes the standard for reward engineering.

6 Monitoring Rewards

In this section, we define monitoring rewards, an extension

of temporal rewards (Bacchus, Boutilier, and Grove 1996;

Thiébaux et al. 2006; Littman et al. 2017; Brafman, De Giaco-

mo, and Patrizi 2018) based on the four satisfaction condi-

tions from runtime monitoring on finite traces (De Giaco-

como et al. 2014).

A monitoring reward is a 5-tuple $(\varphi, r, c, s, f)$ where $\varphi$

is a temporal formula and $r, c, s, f$ are integers; we call $\varphi$

the reward formula and $r, c, s, f$ the reward values. When

a monitoring reward of such kind is specified, an agent re-

ceives a reward value $r$ (reward) when $\varphi$ is temporarily true

in the current partial trace, $c$ (cost) when it is temporarily

false, $s$ (success) when permanently true, and $f$ (failure)

when permanently false. We call each of the former cases a

reward condition. If not stated otherwise, we assume that

$r \geq 0, s \geq 0, c \leq 0$ and $f \leq 0$, as we consider this the nat-

ural interpretation of the four conditions. If multiple mon-

itoring rewards are given at the same time, then the agent

receives the sum of the values computed for each monitoring

reward as specified above.

Example 1. Consider the monitoring reward:

$\langle (\Box \neg p) \lor (\Diamond q), 1, -1, 10, 0 \rangle$.

If $p$ does not hold anytime in the current trace, and the same

holds true for $q$, then the agent receives reward $1$. If $p$

does not hold sometimes in the current trace, and the same for

$q$, then the agent receives $-1$. If $1$ holds sometimes in the

current trace, then the agent receives $10$.

We have that exactly one of the reward conditions is true

at any moment, because a formula is either temporarily true,

temporarily false, permanently true, or permanently false in

a trace.

Theorem 9. $\pi \models [\varphi = T]$ holds for exactly one $T$ from

\{temp_true, temp_false, perm_true, perm_false\}.

Proof. We have that $\pi \models [\varphi = T]$ implies that either $\pi \models [\varphi = temp_true]$ or $\pi \models [\varphi = perm_true]$, and similarly for the dual case $\pi \not\models [\varphi]$. Then, the theorem follows from the fact that either $\pi \models \varphi$ or $\pi \not\models \varphi$.

When the reward condition is permanently true (or false)
in the current trace, the agent will keep receiving the same

reward value. In fact, the reward condition will be perma-
nently true (resp., false) at any future step, and in particular

will not become temporarily true or false.
Theorem 10. For \( T \in \{ \text{perm\_true}, \text{perm\_false} \} \), we have that \( \pi \models \langle \varphi = T \rangle \) implies \( \pi \pi' \models \langle \varphi = T \rangle \) for every trace extension \( \pi' \) and hence \( \pi \pi' \not\models \langle \varphi = T' \rangle \) for \( T' \in \{ \text{temp\_true}, \text{temp\_false} \} \).

As a consequence of the previous theorem, if are interested in traces of a fixed length (e.g., episodes in RL), the total reward value on a trace can be computed as soon as the reward condition becomes permanently true or false.

Each monitoring reward \( \langle \varphi, r, c, s, f \rangle \) admits the equivalent dual form \( -\varphi, c, r, f, s \), where we negate the formula and swap values for reward and cost, and for success and failure. To see the equivalence, it suffices to notice that a \( \varphi \) is temporarily/permanently true iff \( -\varphi \) is temporarily/permanently false.

Theorem 11. Monitoring rewards \( \langle \varphi, r, c, s, f \rangle \) and \( -\varphi, c, r, f, s \) return the same value on every trace.

Monitoring rewards capture \( \text{LDL}_f / \text{LTL}_f \) specifications as in (Brafman, De Giacomo, and Patrizi 2018). Specifically, a specification \( \langle \varphi, r \rangle \) can be restated as the monitoring reward \( \langle \varphi, r, 0, r, 0 \rangle \). What is less obvious is that each monitoring reward can be expressed as a set of four \( \text{LDL}_f \) specifications. In fact, the four reward conditions can be directly expressed in \( \text{LDL}_f \) without any meta-logical machinery (De Giacomo et al. 2014). So a reward \( \langle \varphi, r, c, s, f \rangle \) can be restated as a set of specifications \( \{ (\varphi^+, r), (\varphi^-, c)(\varphi^0, s), (\varphi^1, f) \} \).

Since \( \text{LDL}_f \) rewards capturing monitoring rewards, we can turn an NMRDP with monitoring rewards into an equivalent MDP using the extended MDP construction based on DFAs (Brafman, De Giacomo, and Patrizi 2018). We argue that this construction introduces an unnecessary state overhead in the case of monitoring rewards.

The formulas \( \varphi^+, \varphi^-, \varphi^0, \varphi^1 \) (and also \( \varphi \)) admit shape-equivalent DFAs (De Giacomo et al. 2014). Their corresponding Moore reward machines can be combined into a single machine by taking the direct sum—see Section 3. In particular, the resulting reward machine has the same number of states of the DFA for \( \varphi \), and hence we have the same state overhead of a simple reward specification \( \langle \varphi, r, c \rangle \).

Example 2. Consider the monitoring reward:

\[ \langle \bullet (a U b), r, c, s, f \rangle. \]

In Figure 5 we show the equivalent Moore reward machine. This is the result of a direct sum between 4 Moore machines, where each of them models one condition at a time. The conditions are highlighted with different colors per state.

As a result, monitoring rewards introduce no additional state overhead compared to simple temporal rewards.

Theorem 12. If an NMRDP \( \langle S, A, Tr, \langle \varphi, r, c \rangle \rangle \) admits an equivalent MDP which introduces a state overhead \( n \), then every NMRDP of the form \( \langle S, A, Tr, \langle \varphi, r, c, s, f \rangle \rangle \) admits an equivalent MDP which introduces state overhead \( n \).

---

**7 Applications in RL**

One field that can benefit from the approach described is Reinforcement Learning. Indeed, most RL algorithms assume the underlying hidden model to be an MDP. Hence, the approach described in the previous sections can be very useful for RL. The idea is that we can give a specification at a high level of abstraction on how to give rewards. The rewards are then given by the induced reward transducer, as explained earlier in this work. Moreover, being the overhead the smallest possible, algorithms on such MDPs are more effective.

Reward engineering is a very crucial task when devising RL domains. The specification of reward functions can be cumbersome and error-prone, breaking RL algorithms in surprising, counterintuitive ways. This phenomenon is known in the community as reward hacking (Amodei et al. 2016). An illustrative example is shown in (Clark and Amodei 2016). No, we have that the experiment designer can specify monitoring of temporal specification to have a finer control on the reward given to the agent, despite having a concise, human-friendly language like \( \text{LTL}_f / \text{LDL}_f \).

We argue that is more convenient to think in terms of monitoring rewards of the form \( \langle \varphi, r, c, s, f \rangle \) than temporal specifications of the form \( \langle \varphi, r \rangle \).

In Figure 6 is depicted the scenario we have in mind. The agent acts in an environment and observes some features to take the next action. Each observation is passed to a monitor that interpret the observation, updates its state and produces a reward signal that is then given to the agent. In this way, the agent’s behaviour is implicitly driven by the monitor via rewards, specified at high-level by the designer. In the rest of this section, we will describe potential applications of our approach.

**Mountain Car.** The Mountain Car environment (Moore 1991) is a classic RL problem. The state space is the set of pair \( \langle \text{position}, \text{velocity} \rangle \). A reward of \(-1\) is given at each timestep. The goal state is when \( \text{position} \geq 0.5 \). We model the reward function with a monitoring temporal specification \( \langle \Diamond \text{goal}, 0, -1, 0, 0 \rangle \), where \( \text{goal} \) is a fluent that is true when \( \text{position} \geq 0.5 \), false otherwise. The training is performed over the extended MDP, where the state space is the cross product between the original MDP state space and
the Mealy reward machine, shown in Figure 7a. Notice that the reward assignment is completely handled by the framework, according to the current simulation of the machine, in a given episode. Specifications of the form $\diamond p$, when $p$ is a state formula, are useful to capture achievement goals, i.e., a condition that must be satisfied in the future, before the end of the trace.

**Cart Pole.** In the Cart Pole environment (Barto, Sutton, and Anderson 1983), the goal is to prevent a pendulum from falling over. The state space is the set of tuples $\langle position, velocity, pole\_angle, pole\_velocity\rangle$. A reward signal of +1 is given at each time step. The failure states are the ones when either $|pole\_angle| \geq 12^\circ$ or $|position| \geq 2.4$. We model the reward function with a monitoring temporal specification $\langle \Box q, +1, 0, 0, 0 \rangle$. The associated Mealy reward machine is shown in Figure 7b. Specifications of the form $\Box q$, when $q$ is a state formula, are useful to capture maintenance goals, i.e., a condition that must be satisfied until the end of the trace.

**Cliff Walking.** An example of a task which is both achievement and maintenance is the approach-avoid task, expressed by the formula $\neg q U goal$. An instance of such task is the Cliff Walking environment, Ch. 6 of (Sutton and Barto 2018). In this kind of gridworld, the reward is -1 on all transitions except those into a special region at the bottom of the grid, representing “The Cliff”. Stepping into this region incurs a reward of -100 and makes the simulation to fail. The goal is to reach a specific goal state, whereas the cliff region constitutes the set of failure states. Such reward function can be captured by the monitoring specification $\langle \neg cliff U goal, 0, -1, +1, -100 \rangle$. Other examples of environments that can be modeled in the same way are Frozen Lake, the 4x3 world (Ch. 21 of (Russell and Norvig 2010)), and WaterWorld domain (Karthikeyan 2015).

**Taxi domain.** In the Taxi domain (Dietterich 2000) there are 4 locations and the goal is to pick up the passenger at one location (the taxi itself is a possible location) and drop him off in another. The reward is +20 points for a successful drop-off, and -1 point for every timestep it takes. There is also a -10 reward signal for illegal pick-up and drop-off actions. The goal state is when the passenger is dropped off at the right place. We can model the Taxi problem as a sequence task: $\langle \Diamond (p \land q), 0, -1, +20, 0 \rangle$, where $p$ means “pick up the passenger” and $q$ means “drop-off the passenger to the right location”. The bad action penalty is another temporal specification $\langle \Box bad\_action, -10, 0, 0, 0 \rangle$. Although we use two temporal specification, we remind that both get compiled into a more compact single Mealy reward machine. Other RL environments that have sequential tasks are the Minecraft environment (Andreas, Klein, and Levine 2017), the task to break columns in order in Breakout or to visit colors in Sapientino (De Giacomo et al. 2019).

8 Conclusions

In this work we have formalised the notion of overhead as the state extension introduced to describe an NMRDP in the form of an MDP. We have considered the overhead introduced by approaches that directly use the DFAs for the reward formulas (Brafman, De Giacomo, and Patrizi 2018), and argued that part of that overhead is unnecessary if we are not interested to know which reward specifications are accountable for the rewards assigned at any given moment. We have shown that, giving up that information, approaches based on reward machines can build exponentially (in fact factorially) smaller extended MDPs, while never doing worse than direct use of DFAs. We have argued that the temporal logics LTLf are an appropriate language to specify rewards, and then extended temporal specifications to monitoring specifications, which build on the four classic monitoring conditions allowing a reward designer to assign rewards based on temporary/permanent satisfaction of a temporal formula. We have shown how transducer-based approaches allow for implementing monitoring specifications at no extra cost compared to reward specifications; in other words, the extension from one condition to four conditions comes for free. Finally, we have applied monitoring rewards to reinforcement learning, showing how they can be used to capture the reward functions of popular RL environments.

---

4https://gym.openai.com/envs/FrozenLake-v0/
Acknowledgments
Work supported in part by European Research Council under the European Union’s Horizon 2020 Programme through the ERC Advanced Grant WhiteMech (No. 834228).

References

