Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020)
Special Session on KR and Machine Learning

Temporal Logic Monitoring Rewards via Transducers

Giuseppe De Giacomo , Marco Favorito , Luca Iocchi , Fabio Patrizi , Alessandro Ronca
DIAG, Università di Roma "La Sapienza", Italy
{degiacomo, favorito, iocchi, patrizi, ronca}@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 (NMRDPs) 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 (NMRDP) have been proposed (Bacchus, Boutilier, and Grove 1996; Thiébaux 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 (Toro Icarte et al. 2018; Camacho et al. 2019; Toro Icarte et al. 2019), the authors introduce the concept of *reward machine*, an automata-based formalism to encode non-Markovian rewards. 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 $\text{LTL}_f/\text{LDL}_f$ reward specifications and show that these four

conditions can be monitored at no additional cost w.r.t. to satisfaction only, through the use of transducers. Finally, we discuss the use of such kind of LTL_f/LDL_f-based reward specifications in reinforcement learning of non-Markovian specifications.

2 Background

MDPs and RL. A Markov Decision Process (MDP) $\mathcal{M} = \langle S, A, Tr, R\rangle$ contains a set S of states, a set A of actions, a transition function $Tr : S \times A \to Prob(S)$ that returns for every state s and action a a distribution over the next state, and a reward function $R : S \times A \to \mathbb{R}$ 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 $\mathcal{P}$ 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 ρ at state s, denoted $v^\rho(s)$, is the expected sum of (possibly discounted by a factor γ, with $0 \leq \gamma \leq 1$) rewards when starting at state s and selecting actions based on ρ. 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 ρ^*. In discounted cumulative settings, there exists an optimal policy that is *Markovian* $\rho : S \to A$, i.e., ρ 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 $\mathcal{A} = \langle Q, \Sigma, q_0, F, \delta\rangle$ where Q is the (non-empty) finite set of states, Σ is the set of input symbols, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states, and $\delta : Q \times \Sigma \to Q$ is the transition function. The *extended transition function* δ^* of $\mathcal{A}$ is $\delta^*(q, \varepsilon) = q$ and $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$. Automaton $\mathcal{A}$ *accepts* a word w if $\delta^*(q_0, w) \in F$. The *language of* $\mathcal{A}$, written $\mathcal{L}(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts.

Two fundamental kinds of transducers are *Moore machines* and *Mealy machines*. A Moore machine (Moore 1956) is a tuple $\mathcal{M}_o = \langle Q, \Sigma, \Gamma, q_0, \delta, \theta\rangle$ where Q is the set of states, Σ is the set of the input symbols, Γ is the set of the output symbols, q_0 is the initial state, $\delta : Q \times \Sigma \to Q$ is the transition function, and $\theta : Q \to \Gamma$ is the output function that maps states to output symbols. The *output* of $\mathcal{M}_o$ on word $a_1 \dots a_n$ is $\theta(q_0)\,\theta(\delta^*(q_0, a_1)) \dots \theta(\delta^*(q_0, a_1, \dots, a_n))$. A Mealy machine $\mathcal{M}_e$ (Mealy 1955) is like a Moore machine except that its output function $\theta : Q \times \Sigma \to \Gamma$ maps transitions to output symbols, instead of states. Hence the output of $\mathcal{M}_e$ on word $a_1 \dots a_n$ is $\theta(q_0, a_1)\,\theta(\delta^*(q_0, a_1), a_2) \dots \theta(\delta^*(q_0, a_1, \dots, a_{n-1}), a_n)$. Note that, for a Moore machine and a Mealy machine performing the same number of transitions, the output of a Mealy has one symbol less. A Moore/Mealy machine $\mathcal{M}$ corresponds to the *transduction function* $\mathcal{F}_\mathcal{M} : \Sigma^* \to \Gamma^*$ that maps its input to its output. That is, such machines translate words on the input alphabet Σ to words on the output alphabet Γ. 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_f/LDL_f. The logic LTL_f is the classical linear time logic LTL (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 $\mathcal{P}$ of propositional atoms, LTL_f formulas φ are defined as follows:

$$\varphi \quad ::= \quad \phi \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2$$

where ϕ is a propositional formula over $\mathcal{P}$, $\bigcirc$ is the *next* operator and $\mathcal{U}$ is the *until* operator. We use the standard abbreviations: $\varphi_1 \vee \varphi_2 \doteq \neg(\neg\varphi_1 \wedge \neg\varphi_2)$; *eventually* as $\Diamond\varphi \doteq \mathit{true}\,\mathcal{U}\,\varphi$; *always* as $\Box\varphi \doteq \neg\Diamond\neg\varphi$; weak next $\bullet\varphi \doteq \neg\bigcirc\neg\varphi$ (note that on finite traces $\neg\bigcirc\varphi \not\equiv \bigcirc\neg\varphi$); and $\mathit{last} \doteq \bullet\mathit{false}$ denoting the end of the trace. LTL_f is as expressive as first-order logic over finite traces and star-free regular expressions.

LDL_f 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). Here we adopt the version that allows for the empty trace (Brafman, De Giacomo, and Patrizi 2018). An LDL_f formula φ is built as follows:

$$\begin{array}{lcl} \varphi & ::= & \mathit{tt} \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\varrho\rangle\varphi \\ \varrho & ::= & \phi \mid \varphi? \mid \varrho_1 + \varrho_2 \mid \varrho_1; \varrho_2 \mid \varrho^* \end{array}$$

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

A remarkable property of LTL_f/LDL_f is that, for each formula φ, we can construct a DFA $\mathcal{A}_\varphi$ that tracks satisfaction

of φ: $\mathcal{A}_\varphi$ accepts a finite trace π *iff* π satisfies φ. Our results crucially rely on the existence of such an automaton. However, this is not unique to LTL_f/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.(Bacchus, Boutilier, and Grove 1996; Thiébaux et al. 2006; Slaney 2005; Gretton 2007; Gretton 2014; Lacerda, Parker, and Hawes 2014; Lacerda, Parker, and Hawes 2015).

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

Specifying a non-Markovian reward function explicitly is cumbersome and unintuitive, even if only a finite number of traces are to be rewarded. LTL_f/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 $\bar{R}$ using a set of pairs $\{(\varphi_i, r_i)\}_{i=1}^m$, where each φ_i is an LTL_f/LDL_f formula over the propositions $\mathcal{P}$ that selects the traces to reward, and r_i the reward assigned to those traces. When the current (partial) trace is $\pi = \langle s_0, a_1, \ldots, s_{n-1}, a_n\rangle$, the agent receives at s_n each reward r_i whose formula φ_i is satisfied by π.

From NMRDPs to MDPs. In (Brafman, De Giacomo, and Patrizi 2018) it is shown that for any NMRDP $M = \langle S, A, Tr, \{(\varphi_i, r_i)\}_{i=1}^m\rangle$, with φ_i being LTL_f/LDL_f formulas, there exists an MDP $M' = \langle S', A, Tr', R'\rangle$ 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 the same transition probabilities, and corresponding traces have the same rewards (Bacchus, Boutilier, and Grove 1996). Denoting with $\mathcal{A}_{\varphi_i} = \langle Q_i, 2^{\mathcal{P}} \times A, q_{i0}, \delta_i, F_i\rangle$ (notice that $S \subseteq 2^{\mathcal{P}}$ and δ_i is total) the DFA associated with φ_i, the equivalent MDP M' is as follows:

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

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

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

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

Observe that the state space of M' is the product of the state spaces of M and $\mathcal{A}_{\varphi_i}$, and that the reward R' is Markovian. In other words, the (stateful) structure of the LTL_f/LDL_f formulas φ_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 $M = \langle S, A, Tr, \{(\varphi_i, r_i)\}_{i=1}^m\rangle$ is equivalent to the MDP $M' = \langle S', A, Tr', R'\rangle$ defined above.*

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

Lemma 1 ((Brafman, De Giacomo, and Patrizi 2018))**.** *Given an NMRDP M and an equivalent MDP M', every policy ρ' for M' has an equivalent policy $\bar{\rho}$ for M and vice versa.*

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_f/LDL_f rewards over an NMRDP $M = \langle S, A, Tr, \{(\varphi_i, r_i)\}_{i=1}^m\rangle$, with Tr and $\{(\varphi_i, r_i)\}_{i=1}^m$ hidden to the learning agent can be reduced to RL over the MDP $M' = \langle S', A, Tr', R'\rangle$ 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; Ly et al. 2013; De Giacomo et al. 2014). Given an LTL_f/LDL_f formula φ and a trace π, we say that:

- φ is *temporarily true* in π if π satisfies φ and there is a continuation of π that does not satisfy φ, and we write $\pi \models [\![\varphi = temp\_true]\!]$;
- φ is *temporarily false* in π if π does not satisfy φ and there is a continuation of π that satisfies φ, and we write $\pi \models [\![\varphi = temp\_false]\!]$;
- φ is *permanently true* in π if π and all its continuations satisfy φ, and we write $\pi \models [\![\varphi = perm\_true]\!]$;
- φ is *permanently false* in π if π and all its continuations do not satisfy φ, and we write $\pi \models [\![\varphi = perm\_false]\!]$.

3 Reward Transducers

In the literature, several approaches have been proposed to automatically extend MDPs so to capture non-Markovian or temporally-extended rewards. The central idea is to extend states with sufficient information from the past. In (Bacchus, Boutilier, and Grove 1996; Thiébaux et al. 2006) rewards are expressed by using variants of LTL, and states are extended by suitably annotating them exploiting the structure of the reward formulas. In (Littman 2015; Littman et al. 2017) the necessity of a declarative mechanism for expressing complex rewards was again brought about to tame the difficulty of reward engineering in complex systems. In (Brafman, De Giacomo, and Patrizi 2018) rewards where expressed using LTL_f/LDL_f temporal logic on finite traces, which are first translated into DFAs, and then the extended MDP is obtained as the cross product of such DFAs with the original MDP—as discussed in Section 2. This approach is applied to reinforcement learning in (De Giacomo et al. 2019).

In (Toro Icarte et al. 2018; Camacho et al. 2019) the idea of handling non-Markovian rewards through a finite machine (as a DFA) is decoupled from where the machine comes from (e.g., from an LTL_f specification) and the focus becomes the machine itself, called "reward machine".

In this paper we first focus on reward machines directly as in (Toro Icarte et al. 2018; Camacho et al. 2019), introducing the general notion of *reward transducers*; then later we will show some advanced way of declaratively specifying such transducers that extend the ideas in (Bacchus, Boutilier, and Grove 1996; Thiébaux et al. 2006; Brafman, De Giacomo, and Patrizi 2018).

In our context, a reward transducer maps MDP traces of the form $\pi = \langle (s_0, a_1), (s_1, a_2), \ldots, (s_{n-1}, a_n) \rangle$ 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 $\mathcal{R} \subset \mathbb{R}$. 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 $\bar{R}$ as a transduction function and we can implement it as a transducer. We observe that a temporal specification (φ, r) can be transformed into an equivalent reward transducer. Indeed, we can transform the associated DFA $\mathcal{A}_\varphi$ into the Moore machine $\mathcal{M}_o^\varphi = \langle Q, 2^\mathcal{P} \times A, \{0, r\}, q_0, \delta, \theta_o \rangle$ where Q, q_0 and δ are defined as in $\mathcal{A}_\varphi$, and $\theta_o(q) = 0$ if $q \notin F$, $\theta_o(q) = r$ otherwise. That is, the Moore machine outputs 0 for every prefix that does not satisfy φ and outputs r for every prefix that satisfies it. Analogously we can define a Mealy machine $\mathcal{M}_e^\varphi = \langle Q, 2^\mathcal{P} \times A, \{0, r\}, q_0, \delta, \theta_e \rangle$ 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 $\mathcal{M}_o^1 = \langle Q_1, \Sigma, \mathcal{R}_1, q_{10}, \delta_1, \theta_1 \rangle$ and $\mathcal{M}_o^2 = \langle Q_2, \Sigma, \mathcal{R}_2, q_{20}, \delta_2, \theta_2 \rangle$, which is a Moore reward machine outputting the sum of the rewards of the two initial machines. Formally, the sum machine $\mathcal{M}'_o = \mathcal{M}_o^1 + \mathcal{M}_o^2$ is $\langle Q, \Sigma, \mathcal{R}, q_0, \delta, \theta_o \rangle$ where:

- $Q = Q_1 \times Q_2$;
- $\mathcal{R} = \{r_1 + r_2 \mid r_1 \in \mathcal{R}_1, r_2 \in \mathcal{R}_2\}$;
- $q_0 = (q_{10}, q_{20})$;
- $\delta = \{(q_1, q_2) \xrightarrow{\sigma} (q'_1, q'_2) \mid \forall i \in \{1, 2\}.\ q_i \xrightarrow{\sigma} q'_i \in \delta_i\}$;
- $\theta_o = \{(q_1, q_2) \mapsto r_1 + r_2 \mid \forall i \in \{1, 2\}.\ \theta_i(q_i) = r_i\}$.

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 $\mathcal{M}_o^1 + \mathcal{M}_o^2$ is the sum of the functions of $\mathcal{M}_o^1$ and $\mathcal{M}_o^2$, i.e., $\mathcal{F}_{\mathcal{M}_o^1 + \mathcal{M}_o^2} = \mathcal{F}_{\mathcal{M}_o^1} + \mathcal{F}_{\mathcal{M}_o^2}$. 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 how the output function is built:

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

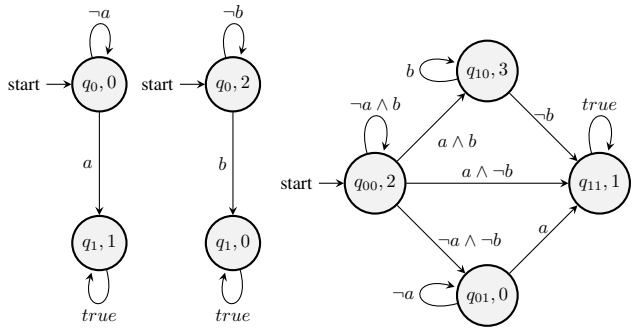
Direct Sum of Reward Transducers. If the two machines are basically the same machine except for the output function, then we can build a sum machine simply by taking the sum of their output function. In this case we call the two machines *shape-equivalent*—a notion inspired by the shape-equivalence for DFAs in (De Giacomo et al. 2020). Specifically, $\mathcal{M}_o^1$ and $\mathcal{M}_o^2$ 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 $\mathcal{M}_o^1 \oplus \mathcal{M}_o^2 = \langle Q, \Sigma, \mathcal{R}, q_0, \delta, \theta \rangle$ where Q, Σ, q_0, and δ are the common states, input alphabet, initial state, and transition function, respectively, $\mathcal{R}$ is defined as for $\mathcal{M}_o^1 \oplus \mathcal{M}_o^2$, and $\theta = \theta_1 + \theta_2$. It is again the case that $\mathcal{F}_{\mathcal{M}_o^1 \oplus \mathcal{M}_o^2} = \mathcal{F}_{\mathcal{M}_o^1} + \mathcal{F}_{\mathcal{M}_o^2}$. 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 $(\Diamond a, +1)$ and $(\Box b, +2)$ and their sum. In Figure 2, we show the equivalent Mealy reward machines.

4 Extending MDPs via Reward Transducers

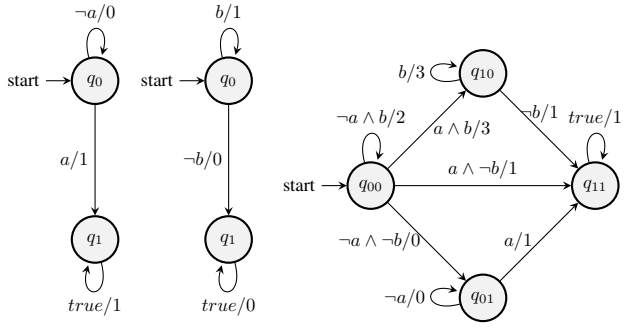
Rewarding complex behaviours is a challenging task, and temporal logic provides the right level of abstraction to address the problem (Littman 2015; Littman et al. 2017). This is the philosophy behind NMRDPs with LTL_f/LDL_f rewards (Brafman, De Giacomo, and Patrizi 2018). NMRDPs can be reduced to MDPs, 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.

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 $|S'| - |S|$.*



(a) $\mathcal{M}_o^1$ (b) $\mathcal{M}_o^2$ (c) $\mathcal{M}'_o = \mathcal{M}_o^1 + \mathcal{M}_o^2$

Figure 1: In Figure (a), the Moore reward machine $\mathcal{M}_o^1$ for the temporal specification $(\Diamond a, +1)$. In Figure (b), Moore reward machine $\mathcal{M}_o^2$ for the temporal specification $(\Box b, +2)$. In Figure (c), the Moore machine of the sum of $\mathcal{M}_o^1$ and $\mathcal{M}_o^2$.



(a) $\mathcal{M}_e^1$ (b) $\mathcal{M}_e^2$ (c) $\mathcal{M}'_e = \mathcal{M}_e^1 + \mathcal{M}_e^2$

Figure 2: Here we show the same specifications depicted in Figure1, but implemented as Mealy reward machines. In Figure (a), the Mealy reward machine $\mathcal{M}_e^1$ for the temporal specification $(\Diamond a, +1)$. In Figure (b), Moore reward machine $\mathcal{M}_e^2$ for the temporal specification $(\Box b, +2)$. In Figure (c), the Mealy machine of the sum of $\mathcal{M}_e^1$ and $\mathcal{M}_e^2$.

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 it 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 $\mathcal{M}_o^i$ be the Moore reward machine for (φ_i, r_i). We define $\mathcal{M}_o = \langle Q_o, 2^{\mathcal{P}} \times A, \mathcal{R}_o, q_0, \delta_o, \theta_o \rangle$ as the Moore reward machine obtained by minimising the sum of all the other Moore reward machines, i.e., $\mathcal{M}_o$ is the minimum machine equivalent to $\mathcal{M}_o^1 + \cdots + \mathcal{M}_o^m$. We derive the new MDP $M_o = \langle S_o, A, Tr_o, R_o \rangle$ as follows:

- $S_o = S \times Q_o$;
- $Tr_o : S_o \times A \times S_o \to [0, 1]$ 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:

$$R_o((q, s), a) = \theta_o(\delta_o(q, (s, a)))$$

We have 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 δ. Finally, notice that $\theta_o(q) = \sum_i \theta_o^i(q_i)$, where $\theta_o^i(q_i) = r_i$ when $q_i \in F_i$, so R_o is simply a compact representation of R'. □

The minimisation step in the construction above is important. Even assuming that the machines $\mathcal{M}_o^i$ are minimum, their sum machine may not be; thus, the minimisation step is required to minimise the state space of the resulting MDP M_o, hence 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 $\mathcal{O}(n)$ on M.*

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

$$\bigcirc^i(\neg p_1 \wedge \cdots \wedge \neg p_{i-1} \wedge p_i \wedge \neg p_{i+1} \wedge \cdots \wedge \neg p_n)$$

and Tr consists of transitions

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

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 φ_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 $\mathcal{M}_o^i$ for $(\varphi_i, 1)$ 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 $\{p_i\}$. The minimum reward machine $\mathcal{M}_o$ equivalent to the sum of the machines $\mathcal{M}_o^i$ has 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 space 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. 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 with its transition function. Hence, from the MDP $M_o = \langle S_o, A, Tr_o, R_o \rangle$, we can construct a new MDP $M_e = \langle S_e, A, Tr_e, R_e \rangle$ where everything is defined as in M_o except $R_e : S_e \times A \to \mathbb{R}$ 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. □

5 Rewards as Temporal Specifications

In this section we argue for the case of the temporal logics LDL_f/LTL_f as an appropriate language to specify rewards. In particular, they capture Markovian rewards without loss of efficiency (using transducers) and they can be more 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 $M = \langle S, A, Tr, R \rangle$. Such an MDP is captured by the following NMRDP:

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

where $\varphi_{(s,a)}$ has the form $\Diamond(s \wedge a \wedge last)$—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 ρ by ignoring the history but the last state. Analogously, from ρ we can define a $\bar{\rho}$ such that for all the possible traces $\pi = \langle (s_0, a_1), \ldots, (s_{n-1}, a_n) \rangle$ that end up in state s_n, we have $\bar{\rho}(s_0, \ldots, s_n) = \rho(s_n)$. □

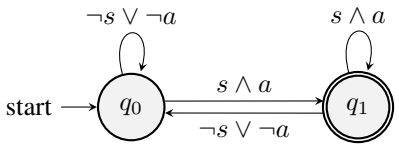


Figure 3: Automaton corresponding to $\varphi_{(s,a)} = \Diamond(s \wedge a \wedge last)$

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

Capturing reward transducers. Temporal specifications capture Moore (and Mealy) reward machines. To see this, consider a Moore reward machine $\mathcal{M}_o = \langle Q, \Sigma, \mathcal{R}, q_0, \delta, \theta_o \rangle$. For each $q \in Q$, let $\mathcal{A}_q$ be the DFA $\langle Q, \Sigma, q_0, \{q\}, \delta \rangle$, and let $\varphi_q = \langle \varrho_q \rangle end$ with ϱ_q a regular expression that captures the language $\mathcal{L}(\mathcal{A}_q)$. Notice that $\mathcal{M}_o$ is captured by the temporal specification $\{(\varphi_q, \theta_o(q))\}_{q \in Q}$. Mealy reward machines can be captured as well, since they can be converted into Moore reward machines. On the other hand, specifications can always be compiled into a reward transducer, as discussed in Section 3. Furthermore, temporal specifications can be more succinct than Moore (and Mealy) reward machines, as shown in the following theorem.

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

Proof. Consider a set A of at least two actions that an agent can perform. In addition, the agent can perform an action e(nd) that marks the end of a sequence of actions, and can observe a c(ommand). We use a construction from (Kupferman and Vardi 2005), adapted to finite traces in (De Giacomo and Rubin 2018). The construction consists of the regular language

$$\mathcal{L}_n = \{(A + e)^* \cdot e \cdot \mathbf{s} \cdot e \cdot (A + e)^* \cdot c \cdot \mathbf{s} \cdot e^+ \mid \mathbf{s} \in A^n\}$$

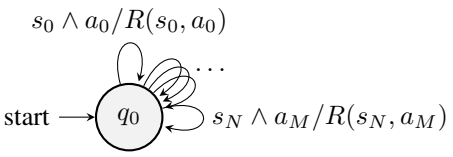


Figure 4: Mealy reward machine that encodes the Markovian reward function R. It has a unique state with a self-loop for each domain value of R.

and of its LTL_f specification

$$\varphi_n = \left(\neg c \mathcal{U} \left(c \wedge \bigwedge_{i=1}^{n} \bigcirc^i \left(\bigvee_{a \in A} a\right) \wedge \bigcirc^{n+1} \Box e\right)\right) \wedge \Diamond \left(e \wedge \bigwedge_{i=1}^{n} \bigvee_{a \in A} \bigcirc^i a \wedge \Box \left(c \rightarrow \bigcirc^i a\right)\right).$$

The construction has two key properties: (*i*) the size of φ_n is $\mathcal{O}(n^2)$, and (*ii*) the size of the minimum DFA for $\mathcal{L}_n$ is $\Omega(2^{2^n})$ (Chandra, Kozen, and Stockmeyer 1981). Consider now the reward function $\bar{R}_n$ that assigns reward 1 to traces in $\mathcal{L}_n$. Intuitively, we reward the agent to initially perform sequences of actions from A with each sequence ended by the action e, and then, upon seeing the command c, perform a sequence $\mathbf{s}$ of length n already performed before. The reward $\bar{R}_n$ is captured by the specification $(\varphi_n, 1)$ which has size $\mathcal{O}(n^2)$, and every reward machine for $\bar{R}_n$ has size $\Omega(2^{2^n})$ since it is at least as big the minimum DFA for $\mathcal{L}_n$. □

Specifying common rewards in logic. It is often the case that we can represent the same transition-based reward function R with much less effort, by specifying a non-Markovian reward fully specified by a much more intelligible LTL_f formula. For example, consider the Mountain Car environment (Moore 1991) 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 *"as long as the car has not reached the goal, give reward -1.0"*. Such statement in natural language can be formalized into a temporal specification, whilst being relatively close in terms of readability to its original formulation. By Theorem 7, we know we can always represent such reward function with temporal specifications. The reward function of that environment can be represented by the specification $(\varphi, -1.0)$ where $\varphi = \neg\Diamond p$ and p means *the car has reached the goal*, a proposition opportunely extracted from the state space. By translating the formula into a DFA $\mathcal{A}_\varphi$ and employing the compilation into the equivalent MDP as explained in Section 2, we make such non-Markovian reward learnable by classic RL algorithms. Thus, we can specify rewards using high-level formal specifications, and then compile them automatically into standard models compatible with solvers.

To pursue the analogy with software engineering: the "raw" R is binary language, the equivalent $\{(\varphi_{(s,a)}, R(s,a))\}_{s \in S, a \in A}$ is the *decompiled* program in a high-level language, say the C++ language, and the 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 based on the four satisfaction conditions from runtime monitoring on finite traces (De Giacomo et al. 2014). They provide an additional layer of abstraction which allows one to focus on one condition φ and to assign different rewards based on how the current trace satisfies φ. Furthermore, in some cases, monitoring rewards allow one to derive the value of future rewards, giving additional guidance to the learning process.

Example 1. *The condition "never p or eventually q" can be temporarily true, temporarily false, or permanently true. We can define a monitoring reward that returns* 1 *when the condition is temporarily true,* -1 *when temporarily false, and* 10 *when permanently true.*

Definition 3. *A* monitoring reward *is a 5-tuple* $\langle \varphi, r, c, s, f \rangle$ *where φ is a temporal formula and r, c, s, f are integers; we call φ the* reward formula *and r, c, s, f the* reward values.

When a monitoring reward of the form above is specified, an agent receives a reward value r (reward) when φ 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 natural interpretation of the four conditions. If multiple monitoring rewards are given at the same time, then the agent receives the sum of the values computed for each monitoring reward. We can now formalise the monitoring reward given in the previous example.

Example 2. *Consider the monitoring reward:*

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

If p does not hold anytime in the current trace, and the same holds for q, then the agent receives reward 1*. If p does hold sometimes in the current trace, and q does not hold anytime, then the agent receives* -1*. If q 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. *$\pi \models [\![\varphi = \mathcal{T}]\!]$ holds for exactly one $\mathcal{T}$ from* $\{temp\_true, temp\_false, perm\_true, perm\_false\}$.

Proof. We have that $\pi \models \varphi$ 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 permanently true (resp., false) at any future step, and in particular will not become temporarily true or false.

Theorem 10. *For $\mathcal{T} \in \{perm\_true, perm\_false\}$, we have that $\pi \models [\![\varphi = \mathcal{T}]\!]$ implies $\pi\pi' \models [\![\varphi = \mathcal{T}]\!]$ for every trace extension π'—and hence $\pi\pi' \not\models [\![\varphi = \mathcal{T}']\!]$ for $\mathcal{T}' \in \{temp\_true, temp\_false\}$.*

As a consequence of the previous theorem, if we 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. This ability can be used to better guide the learning process.

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

Theorem 11. *Monitoring rewards (φ, r, c, s, f) and $(\neg\varphi, c, r, f, s)$ return the same value on every trace.*

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

Since LDL_f rewards capture 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. In fact, the formulas $\varphi^r, \varphi^c, \varphi^s, \varphi^f$ (and also φ) admit shape-equivalent DFAs, as an immediate consequence of Theorem 3 and Corollary 1 of (De Giacomo et al. 2020). Thus, the 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 as the DFA for φ, and hence we have the same state overhead of a simple reward specification (φ, r_φ).

Example 3. *Consider the monitoring reward:*

$$\langle \bullet(a \,\mathcal{U}\, b), r, c, s, f \rangle.$$

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

[1] Note that we need LDL_f and not simply LTL_f because we need to represent prefixes of traces.

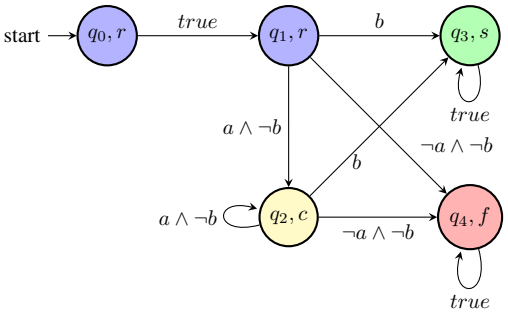


Figure 5: Moore machine for $\langle \bullet(a \,\mathcal{U}\, b), r, c, s, f\rangle$. States are color-coded based on the condition they monitor: purple for $temp\_true$, yellow for $temp\_false$, green for $perm\_true$, and red for $perm\_false$.

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). Here, we propose 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 LTL_f/LDL_f. We argue that is more convenient to think in terms of monitoring rewards of the form (φ, r, c, s, f) than temporal specifications of the form (φ, r).

In Figure 6 is depicted the scenario we have in mind. The world states are described by propositional features. The agent acts in an environment and observes such 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 pairs $\langle position, velocity\rangle$. A reward of -1 is given at each timestep. The goal state is when $position \geq 0.5$. We model the reward function with a monitoring temporal specification $(\Diamond goal, 0, -1, 0, 0)$, where $goal$ is a fluent that is true when $(position \geq 0.5)$, false otherwise. The training is performed over the extended MDP, where the state space is

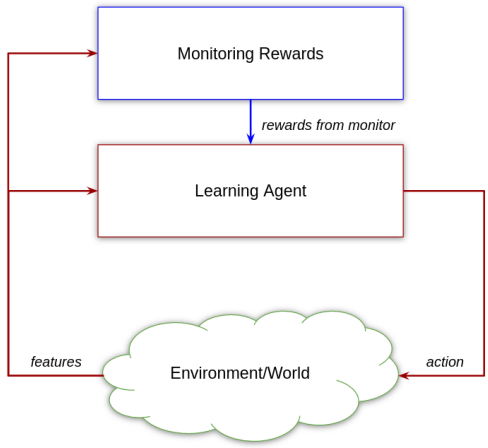
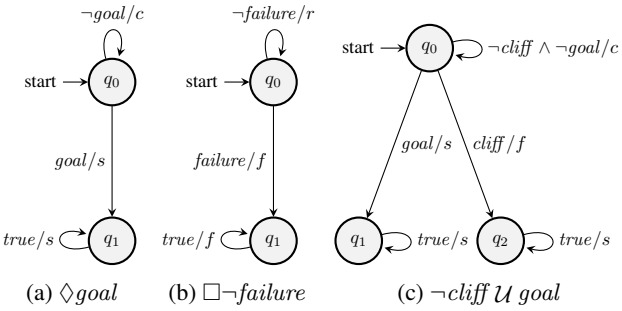


Figure 6: An RL scenario with monitoring rewards.



(a) $\Diamond goal$ (b) $\Box\neg failure$ (c) $\neg cliff\ \mathcal{U}\ goal$

Figure 7: In (a) the Mealy reward machine for $(\Diamond goal, r, c, s, f)$, in (b) the Mealy reward machine for $(\Box\neg failure, r, c, s, f)$, and in (c) the Mealy reward machine for $(\neg cliff\ \mathcal{U}\ goal, r, c, s, f)$.

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 satisfying $|pole\_angle| \geq 12°$ or $|position| \geq 2.4$. We model the reward function with a monitoring temporal specification $(\Box\neg failure, +1, 0, 0, 0)$, where $failure$ is true in failure states. 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., conditions that must be satisfied until the end of the trace.

Cliff Walking. A task that is both achievement and maintenance is the *approach-avoid* task, expressed by the formula $\neg failure\,\mathcal{U}\ goal$. An instance of such task is the Cliff Walking environment in 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 $(\neg cliff\ \mathcal{U}\ goal, 0, -1, +1, -100)$. Other examples of environments that can be modeled in the same way are Frozen Lake (OpenAI 2016), the 4x3 world in Ch. 21 of (Russell and Norvig 2010), and WaterWorld domain (Karpathy 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: $(\Diamond(p \wedge \Diamond q), 0, -1, +20, 0)$, 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 $(\Diamond bad\_action, -10, 0, 0, 0)$. 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 LDL_f/LTL_f 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.

Acknowledgments

Work supported in part by European Research Council under the European Union's Horizon 2020 Programme through the ERC Advanced Grant WhiteMech (N. 834228) and the AI4EU project (N. 825619), and in part by Sapienza University of Rome through the Project DRAPE: Data-awaRe Automatic Process Execution.

References

Alshiekh, M.; Bloem, R.; Ehlers, R.; Könighofer, B.; Niekum, S.; and Topcu, U. 2018. Safe reinforcement learning via shielding. In *AAAI*, 2669–2678.

Amodei, D.; Olah, C.; Steinhardt, J.; Christiano, P. F.; Schulman, J.; and Mané, D. 2016. Concrete problems in AI safety. *CoRR* abs/1606.06565.

Andreas, J.; Klein, D.; and Levine, S. 2017. Modular multitask reinforcement learning with policy sketches. In *ICML*, 166–175.

Bacchus, F.; Boutilier, C.; and Grove, A. 1996. Rewarding behaviors. In *AAAI*, 1160–1167.

Barto, A. G.; Sutton, R. S.; and Anderson, C. W. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. Syst. Man Cybern.* 13:834–846.

Bauer, A.; Leucker, M.; and Schallhart, C. 2010. Comparing LTL semantics for runtime verification. *J. Log. Comput.* 20(3):651–674.

Brafman, R. I.; De Giacomo, G.; and Patrizi, F. 2018. LTLf/LDLf non-Markovian rewards. In *AAAI*, 1771–1778.

Camacho, A.; Chen, O.; Sanner, S.; and McIlraith, S. A. 2017. Decision-making with non-Markovian rewards: From LTL to automata-based reward shaping. In *RLDM*, 279–283.

Camacho, A.; Toro Icarte, R.; Klassen, T. Q.; Valenzano, R. A.; and McIlraith, S. A. 2019. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In *IJCAI*, 6065–6073.

Chandra, A. K.; Kozen, D.; and Stockmeyer, L. J. 1981. Alternation. *J. ACM* 28(1):114–133.

Clark, J., and Amodei, D. 2016. Faulty reward functions in the wild. https://openai.com/blog/faulty-reward-functions/. Accessed: 15-03-2020.

De Giacomo, G., and Rubin, S. 2018. Automata-theoretic foundations of FOND planning for LTLf and LDLf goals. In *IJCAI*, 4729–4735.

De Giacomo, G., and Vardi, M. Y. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, 854–860.

De Giacomo, G., and Vardi, M. Y. 2015. Synthesis for LTL and LDL on finite traces. In *IJCAI*, 1558–1564.

De Giacomo, G., and Vardi, M. Y. 2016. LTLf and LDLf synthesis under partial observability. In *IJCAI*, 1044–1050.

De Giacomo, G.; De Masellis, R.; Grasso, M.; Maggi, F. M.; and Montali, M. 2014. Monitoring business metaconstraints based on LTL and LDL for finite traces. In *BPM*, volume 8659 of *Lecture Notes in Comput. Sci.*, 1–17.

De Giacomo, G.; Iocchi, L.; Favorito, M.; and Patrizi, F. 2019. Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications. In *ICAPS*, 128–136.

De Giacomo, G.; De Masellis, R.; Maggi, F. M.; and Montali, M. 2020. Monitoring constraints and metaconstraints with temporal logics on finite traces. *CoRR* abs/2004.01859.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *J. Artif. Intell. Res.* 13(1):227–303.

Gretton, C. 2007. Gradient-based relational reinforcement learning of temporally extended policies. In *ICAPS*, 168–175.

Gretton, C. 2014. A more expressive behavioral logic for decision-theoretic planning. In *PRICAI*, volume 8862 of *Lecture Notes in Comput. Sci.*, 13–25.

Karpathy, A. 2015. REINFORCEjs: WaterWorld demo. https://cs.stanford.edu/people/karpathy/reinforcejs/waterworld.html. Accessed: 15-03-2020.

Kupferman, O., and Vardi, M. Y. 2005. From linear time to branching time. *ACM Trans. Comput. Log.* 6(2):273–294.

Lacerda, B.; Parker, D.; and Hawes, N. 2014. Optimal and dynamic planning for markov decision processes with co-safe LTL specifications. In *IROS*, 1511–1516.

Lacerda, B.; Parker, D.; and Hawes, N. 2015. Optimal policy generation for partially satisfiable co-safe LTL specifications. In *IJCAI*, 1587–1593.

Linz, P. 2006. *An introduction to formal languages and automata*. Jones and Bartlett Publishers.

Littman, M. L.; Topcu, U.; Fu, J.; Jr., C. L. I.; Wen, M.; and MacGlashan, J. 2017. Environment-independent task specifications via GLTL. *CoRR* abs/1704.04341.

Littman, M. L. 2015. Programming agent via rewards. Invited talk at IJCAI.

Ly, L. T.; Maggi, F. M.; Montali, M.; Rinderle-Ma, S.; and van der Aalst, W. M. P. 2013. A framework for the systematic comparison and evaluation of compliance monitoring approaches. In *EDOC*, 7–16.

Mealy, G. H. 1955. A method for synthesizing sequential circuits. *Bell Syst.* 34:1045–1079.

Moore, E. F. 1956. Gedanken-experiments on sequential machines. *Automata Studies* 129–154.

Moore, A. W. 1991. Variable resolution dynamic programming. In *ML91*, 333–337.

OpenAI. 2016. Frozenlake-v0. https://gym.openai.com/envs/FrozenLake-v0/. Accessed: 30-06-2020.

Pnueli, A. 1977. The temporal logic of programs. In *FOCS*, 46–57.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley.

Quint, E.; Xu, D.; Dogan, H.; Hakguder, Z.; Scott, S.; and Dwyer, M. B. 2019. Formal language constraints for markov decision processes. *CoRR* abs/1910.01074.

Rabin, M. O., and Scott, D. S. 1959. Finite automata and their decision problems. *IBM J. Res. Dev.* 3(2):114–125.

Russell, S. J., and Norvig, P. 2010. *Artificial Intelligence: A Modern Approach*. Pearson Education.

Slaney, J. K. 2005. Semipositive LTL with an uninterpreted past operator. *L. J. IGPL* 13(2):211–229.

Sutton, R. S., and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. A Bradford Book.

Thiébaux, S.; Gretton, C.; Slaney, J. K.; Price, D.; and Kabanza, F. 2006. Decision-theoretic planning with non-Markovian rewards. *J. Artif. Intell. Res.* 25:17–74.

Toro Icarte, R.; Klassen, T.; Valenzano, R.; and McIlraith, S. 2018. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *ICML*, 2107–2116.

Toro Icarte, R.; Waldie, E.; Klassen, T.; Valenzano, R.; Castro, M.; and McIlraith, S. 2019. Learning reward machines for partially observable reinforcement learning. In *NIPS*, 15523–15534.