Regular Clocks for Temporal Task Specifications in Reinforcement Learning

Giuseppe De Giacomo[1,2][0000−0001−9680−7658], Marco Favorito[3][0000−0001−9566−3576], and Fabio Patrizi[2][0000−0002−9116−251X]

[1] University of Oxford, UK
[2] Sapienza University of Rome, Italy
{degiacomo,patrizi}@diag.uniroma1.it
[3] Bank of Italy
marco.favorito@bancaditalia.it

Abstract. Several recent approaches in reinforcement learning are studying a conceptual architecture where the environment is simultaneously represented at two (or more) levels of abstraction, with the environment providing two traces of data/events/features/fluents, one at a lower-level/finer grain and one at a higher-level/coarser grain. For simplicity, most of this literature assumes that the instants of the two traces match. In this paper, we drop this strong assumption and introduce an explicit mapping between the low-level and the high-level traces that the high-level trace perceives as a clock defined in terms of properties of segments of the low-level one. We investigate the case of regular mappings, where the segments that induce clock ticks are specified by a regular language property or a finite-state machine. We show that if both the clock and the high-level specifications are expressed as finite-state machines, such as reward machines, we can combine the two specifications in polynomial time into a single machine incorporating the clock. We then investigate the case in which both the clock and the high-level task are specified declaratively, e.g., in linear temporal logics on finite traces such as LTL_f and LDL_f, and show that this yields a notable representational advantage wrt a flattened representation where the clock is not explicit.

Keywords: Clock specification· Reinforcement Learning · Temporal Tasks

1 Introduction

Several recent works are focusing on a conceptual architecture where the environment is simultaneously represented at two (or more) levels of abstraction, each providing a different trace (or traces) of data/events/features/fluents: one at a lower-level/finer grain and one at a higher-level/coarser grain [16, 18, 2, 5, 28, 3, 6, 15]. For example, the low-level trace could include environment features directly observed by a reinforcement-learning (RL) agent while the high-level trace could include logical fluents observed by a KR(-based) monitor, such as a temporal specification, and used to reward the RL agent for carrying out some task, according to the fulfilment of the (high-level) specification.

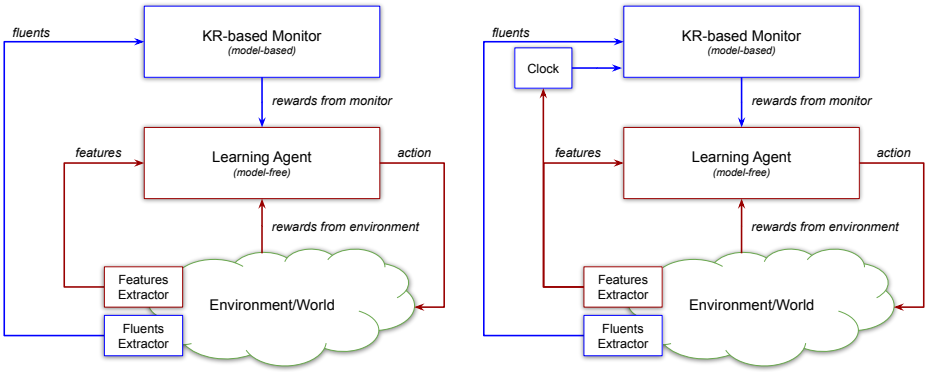
For concreteness, we consider the unclocked setting discussed in [5], depicted in Fig. 1a (the KR monitor was called "restraining bolt" in 1a). As standard in RL, the RL agent interacts with the environment by observing a number of features extracted by a suitable module, e.g., a set of sensors, performing some actions, and possibly obtaining rewards. The observed features produce the low-level trace. Besides, there are additional properties of the environment, called *fluents*, that the RL agent is, in general, unaware of but are observable to an external KR monitor. These correspond to the high-level trace. Fluents can be complex properties which depend on the features but can also be features themselves, possibly inaccessible to the agent due, e.g., to a lack of suitable sensors. The KR monitor requires the agent to fulfil some requirements or carry out an additional task, wrt to that implicitly defined by the standard reward function. This is achieved by defining an additional, possibly non-Markovian reward function, based on the fluents, implemented by the KR monitor. It is important to observe that this setting implicitly makes the assumption that the feature and fluent traces are aligned, i.e., they produce the next observation at the same time, [5, 15].

In this paper, we advocate the introduction of a specific component to allow for loosening the synchronicity requirement of the two traces; see Fig 1b. The clock component generates the time points of the high-level trace for the KR monitor by checking relevant properties of the current prefix of the low-level trace of features. This allows for a better representation of the KR monitor, decoupling the handling of the clock from the high-level specification that uses it. For example, imagine that the high-level property checked by the KR monitor depends only on the data items produced at even time points $(0, 2, 4, \ldots)$. Introducing a clock component allows the KR monitor to offer a reward based only on the time points of interest without needing to keep track of each time point's parity; on the other hand, if the clock module is not present, the KR module must track parity, cluttering the specification of the KR monitor itself.

In this paper, we explore the benefits and the implications of dropping the strong *common-clock* assumption for feature and fluent extractors. We do so by introducing an explicit mapping between the low-level and the high-level traces, which is perceived by the high-level trace as a clock, defined in terms of properties of segments of the low-level trace. We require the mapping to be regular, in the sense of regular languages [13].

We study the case in which the KR monitor consists of a Reward Machine (RM) [15], and the clock consists of a finite state transducer, or automaton, (FSA). We solve reinforcement learning in this setting by showing how to compile the clock aware in a more involved reward machine which although clattered by the handling of the clock and hence less intuitive, can be computed automatically in polynomial time. In this way, we obtain both the representational advantage of decoupling the clock from the KR monitor while still maintaining the effectiveness of the reward machine approach.

We then discuss the case where the clock and KR monitor are specified declaratively using a linear temporal logics on finite traces, in particular, LTL_f



(a) Unclocked framework [5]. (b) Clocked framework.

Fig. 1: The standard agent-environment systems with a KR-based monitor.

or LDL_f [8]. Notice that LDL_f has exactly the expressive power of regular expression (i.e., that of Monodic Second Order Logic on finite traces). Instead LTL_f has the expressive power of star-free regular expressions (i.e., that of First-Order Logic on finite traces). As a result, we can compile $\text{LTL}_f/\text{LDL}_f$ KR monitor specifications and clock specifications in Reward Machines and FSA specifications, respectively, and adopt the techniques above for doing reinforcement learning. This gives us a procedure that is worst case 2EXPTIME-complete, as in the case of unclocked specifications [2, 5]. In fact, the overhead introduced by the clock is minimal.

Finally, a natural question arises: can, at least in principle, clocked specifications in LDL_f and LTL_f be translated into unclocked specifications in LDL_f and LTL_f, respectively? In the case of LDL_f, the answer is obviously positive since LDL_f can capture any regular language and hence also that obtained from compiling the clock into a finite state reward machine. For LTL_f, proving that this is the case is not as simple because it has to show that the specific Cartesian Product construction that we use to compile away the clock specification preserves being star-free. We do show this in the paper.

Note that these expressivity results do not induce an easy (polynomial) way of compiling away the logical specification of the clock into the logical specification of the KR monitor. The specific constructions used for the proof would generate a 2EXPTIME-blowup in the specification. We leave it to future work whether this upper-bound can be improved. In any case, as we show here, the approach does not need this compilation.

2 Preliminaries

LTL_f and LDL_f. LTL_f and LDL_f are, respectively, Linear Temporal Logic and Linear Dynamic Logic with finite trace semantics, proposed in [8]. LTL_f

shares the same syntax of LTL [22]. It is as expressive as First-Order Logic over finite traces (FOL) or star-free regular expressions, so strictly less expressive than regular expressions, which, in turn, are as expressive as Monadic Second-Order logic over finite traces (MSO). The semantics of such logic formalisms are given in terms of finite traces denoting a finite, possibly empty, sequence $\pi = \pi_0, \ldots, \pi_n$ of elements from the alphabet $2^{\mathcal{P}}$, containing all possible propositional interpretations of the propositional symbols in $\mathcal{P}$. Notice that, differently from [8], we allow the empty trace as in [2] and [7]. Given a set $\mathcal{P}$ of propositional symbols, LTL_f formulae are built as follows:

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

where tt is the tautology (not to be confused with $true = \phi \vee \neg\phi$), ϕ is a propositional formula over $\mathcal{P}$, $\bigcirc$ is the *next* operator, and $\mathcal{U}$ is the until operator. We adopt the usual Boolean abbreviations for disjunction, implication, etc. In addition, we use common abbreviations of temporal operators. For the *weak next* operator $\bullet$, we have $\bullet\varphi \equiv \neg\bigcirc\neg\varphi$ (notice that in the finite trace case $\neg\bigcirc\neg\varphi \neq \bigcirc\varphi$), for the *release* operator $\mathcal{R}$, we have $\varphi_1 \,\mathcal{R}\, \varphi_2 \equiv \neg(\neg\varphi_1 \,\mathcal{U}\, \neg\varphi_2)$, for *eventually* ($\Diamond$) we have $\Diamond\varphi \equiv true\,\mathcal{U}\,\varphi$, for *always* ($\Box$) we have $\Box\varphi \equiv \neg\Diamond\neg\varphi$. Finally, we have $last \equiv \bullet(false)$.

LDL_f is a temporal logic as natural as LTL_f, but with the full expressive power of Monadic Second-Order logic over finite traces. LDL_f is obtained by merging LTL_f with regular expressions (RE_f) through the syntax of the well-know logic of programs PDL, *Propositional Dynamic Logic* [10, 11], but adopting a semantics based on finite traces. LDL_f is an adaptation of LDL introduced in [27], which, like LTL, is interpreted over infinite traces. We omit the details on the syntax of LDL_f due to lack of space, but one property that we will use is that regular expressions can easily be encoded into a LDL_f formula. The semantics of a $\text{LTL}_f/\text{LDL}_f$ formula is defined over finite traces; its full definition can be found in [2]. Given a finite (possibly empty) trace π, by $\pi, i \models \varphi$ we denote that the $\text{LTL}_f/\text{LDL}_f$ formula φ is satisfied by π at instant $i \in \mathbb{N}$. We write $\pi \models \varphi$, if $\pi, 0 \models \varphi$ and say that π *satisfies* φ. Moreover, from an $\text{LTL}_f/\text{LDL}_f$ formula φ, we can compute a DFA $\mathcal{A}_\varphi$ that accepts all and only the traces that satisfy φ [8, 2].

Automata theory A deterministic finite-state automaton (DFA) [24] 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 finite set of input symbols (alphabet), $q_0 \in Q$ 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 $\delta^*(q, \epsilon) = q$ and $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$. An automaton $\mathcal{A}$ accepts a word w if $\delta^*(q_0, w) \in F$. The language of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set of words that $\mathcal{A}$ accepts. A *non-deterministic* finite-state automaton (NFA) is defined in the same way as a DFA, except for δ, which is a relation rather than a function, i.e. $\delta \subseteq Q \times \Sigma \times Q$. A Mealy machine M_e [20] is a 6-tuple $M_e = \langle Q, \Sigma, \Gamma, q_0, \delta, \theta\rangle$ where Q is the finite set of states, Σ is the finite set of input symbols, Γ is the finite set of the output symbols, q_0 is the initial state, $\delta : Q \times \Sigma \to Q$ is the transition function, and $\theta : Q \times \Sigma \to \Gamma$ is the output

function that maps transition to output symbols. A Moore machine M_o [21] is like a Mealy machine except that the output function is defined as $\theta : Q \to \Gamma$, i.e., it maps states to output symbols. The output of M_e on word $a_1 \dots a_n$ is $\theta^*(a_1 \dots a_n) = \theta(q_0, a_1)\theta(\delta^*(q_0, a_1), a_2) \dots \theta(\delta^*(q_0, a_1, \dots, a_{n-1}), a_n)$. An analogous definition exists for M_o. A Mealy/Moore machine M defines a regular *transduction function* $F_M : \Sigma^* \to \Gamma^*$ mapping words over the input alphabet Σ into words over the output alphabet Γ. DFAs and NFAs are known as *acceptors*, while Mealy and Moore machines as *transducers*. From a DFA $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, F \rangle$, we can obtain a Mealy machine $M_{\mathcal{A}} = \langle Q, \Sigma, \Gamma, q_0, \delta, \theta \rangle$, with $\theta(q, a) = \mathit{accept}$ iff $\delta(q', a) \in F$, s.t., for every word $w \in \Sigma^*$, we have that $w \in \mathcal{L}(\mathcal{A})$ iff the last character of $F_{M_{\mathcal{A}}}(w)$ is *accept*. The reverse is possible, too. Likewise, we can transform a Mealy machine into an equivalent Moore machine, and vice versa. See [13, 17].

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 $Tr : S \times A \to \mathit{Prob}(S)$ that returns for every state s and action a a distribution over the next state, and a reward function $R : S \times A \times S \to \mathbb{R}$ that specifies the reward (a real value) received by the agent when transitioning from state s to state s' by applying action a. A solution to an MDP is a function called a *policy*, assigning an action to each state, possibly depending 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 [23]. 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., [26]. A *non-Markovian* reward function [1] is defined as $\bar{R} : (S \times A)^* \to \mathbb{R}$, i.e. a real-valued function over finite state-action sequences. Usually, $\bar{R}$ is specified using a pair (φ, r), where φ is a $\mathrm{LTL}_f/\mathrm{LDL}_f$ formula: if the current (partial) trajectory is $\pi = \langle s_0, a_1, \dots, s_{n-1}, a_n \rangle$, the agent receives at s_n a reward r iff $\pi \models \varphi$ (where $s_i \in 2^{\mathcal{P}}$) [2]. A *Non-Markov Reward Decision Process (NMRDP)* is like an MDP except that the reward function is non-Markovian.

3 Clocked Framework

Let $\mathcal{M}_{ag} = \langle S, A, Tr_{ag}, R_{ag} \rangle$ be an MDP on which the learning agent acts. Let $f_r : \mathcal{L}^* \to \mathcal{R}$ a *high-level reward function*, with $\mathcal{L} = 2^{\mathcal{F}}$ the set of possible fluents' configurations, and $\mathcal{R} \subseteq \mathbb{R}$ a finite set of reward values. Additionally, we consider a *clock function* (or simply *clock*) $f_c : \mathcal{L}^* \to \{0, 1\}$, with 0 meaning "low" state and 1 meaning "high" state. We also say that the clock *ticks on trace* t whenever $f_c(t) = 1$. The role of the clock is to exclude particular fluent observations before giving them as input to the high-level reward function. We assume that both f_r and f_c are *regular* functions over histories, therefore they can be represented by

a finite-state machine formalism (e.g. Mealy machines or DFA). The diagram in Figure 1b depicts at a high level the scenario we have in mind: the inner loop (red) of interaction between the learning agent and the environment is similar to the agent-environment loop of an RL scenario, while the outer loop (blue) starts from the *fluents extractor*, which outputs a high-level representation of the world state in the form of a fluent configuration $\ell \in \mathcal{L}$, passes through the clock function evaluation and, if the history so far makes the clock to be in the "high" state, then the fluents observation is fed to the high-level reward function.

Figure 2 intuitively explains how the filtering mechanism of the clock function f_c works. Circles represent trace timesteps. The bottom trace t has the finest time granularity. The clock function f_c is evaluated on every trace prefix. Let us introduce some notation for traces: for a trace $t = \ell_0, \ldots, \ell_n$, $\mathsf{length}(t) = n + 1$ (or $|t|$) is a positive integer denoting the length of t, $t[i]$ is the i-th step of t (with $0 \leq i < \mathsf{length}(t)$, and $t[i:j]$ be the subtrace $t[i], t[i+1], \ldots, t[j]$ (with $0 \leq i \leq j < \mathsf{length}(t)$). If the trace prefix at some time i makes the formula f_c true (i.e. $F(t[0:i]) = 1$), then the timestep is passed to the evaluation of f_r, and becomes a timestep of the coarser-grained timestep sequence t'. On the other hand, if for some timestep i, the trace prefix up to that timestep does not make f_c to tick, then the configuration at timestep i, i.e. $t[i]$, is ignored at the higher level trace t'.

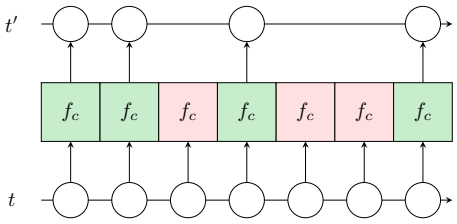


Fig. 2: Intuitive representation of how the clock function f_c projects the low-level trace t into the high-level trace t'.

We now proceed with a complete formalization. To do so, we start with the notion of trace *projection*:

Definition 1 (Trace Projection [9]). *Let $t \in \mathcal{L}^*$ be a trace over the set of fluents configurations $\mathcal{L} = 2^{\mathcal{F}}$, and let f_c be the clock function. The* projection of t onto clock function f_c *is the trace $t|_{f_c} = \ell'_0, \ell'_1, \ldots, \ell'_n$, where $\ell'_i = t[i]$, if $f_c(t[0:i]) = 1$, and $\ell'_i = \epsilon$, otherwise.*

Intuitively, the projection is obtained from t after removing the timesteps with index $i = 0, \ldots, n$ for which the prefix of the trace up to position i (included) does not make f_c evaluate to 1. Note that this notion is analogous to that in [9]. However, the crucial difference is that their clock operator only looks at the current instant, while ours can model temporal constraints.

Example 1. Let $t = \langle \{a, b\}, \{b\}, \{c\}, \{a\}, \{b, c\} \rangle$, and for any trace t', let f_c be a clock function such that $f_c(t') = 1$ if $a \in t[\mathsf{length}(t) - 1]$, otherwise 0. Such behaviour can be intuitively explained as "every time a is true, the clock is high". The projected trace $t|_{f_c}$ is then $\{\{a, b\}, \{a\}\}$. The second, third, and fifth timesteps are filtered out because a does not hold.

Example 2. Let $t = \langle\{a\},\{b\},\{c\},\{a\},\{b\},\{c\}\rangle$, and let $f_c(t') = 1$ if $\mathsf{length}(t')$ mod $2 = 0$, otherwise 0. Intuitively, the clock is high (resp. low) at each even (resp. odd) time step. The projected trace is $t|_{f_c} = \langle\{a\},\{c\},\{b\}\rangle$.

The clock mechanism makes the high-level reward function evaluated only whenever the clock ticks. Given f_c and f_r, the *clocked reward function* $\bar{R}_{cr} : \mathcal{L}^* \to \mathbb{R}$ is as follows:

$$\bar{R}_{cr}(t) = \sum_{i=0}^{|t|-1} \gamma^i f_r(t[0\!:\!i+1]|_{f_c}) \cdot f_c(t[0\!:\!i+1]) \tag{1}$$

The clocked reward function $\bar{R}_{cr}(t)$ gives rewards at the clock tick only, and the reward function f_r is evaluated only on the trace projected onto the clock f_c. Note also that $\bar{R}_{cr}(t)$ is a non-Markovian reward function since its value depends on the full trace history of fluents configurations t.

In our scenario, we are interested in learning an optimal policy for the MDP $\mathcal{M}_{ag}$, where the reward function to optimize is the expected discounted sum of rewards, both from R_{ag} and $\bar{R}_{cr}$. To do so, as in other works, e.g. [5], we assume that the agent actions in A induce a Markovian transition distribution over the features and fluents configuration: $Tr^{\ell}_{ag} : S \times \mathcal{L} \times A \to Prob(S \times \mathcal{L})$, and the interaction between the agent and the environment yields a trajectory $\tau = (s_0, \ell_0), a_0, (s_1, \ell_1), a_1, \ldots, (s_n, \ell_n)$. Hence, the value function of a policy ρ takes the form $v^{\rho}(s_0) = \mathbb{E}_{\tau \sim Tr^{\ell}_{ag}}\big[\bar{R}(\tau)\big]$, where $\bar{R}(\tau) = \sum_{i=0}^{n} \gamma^i R_{ag}(s_i, a_i) + \bar{R}_{cr}(\ell_0, \ldots, \ell_n)$.

Note that, in general, since the reward function is non-markovian, the optimal policy could be non-Markovian too, i.e. $\bar{\rho} : (S \times \mathcal{L})^* \to A$. We call the NMRDP induced by Tr^{ℓ}_{ag} and $\bar{R}(\tau)$ as $\mathcal{M}^{\ell}_{ag}$. We can now state our target problem:

Problem 1. Given the tuple $\langle \mathcal{M}^{\ell}_{ag}, f_r, f_c\rangle$, where $\mathcal{M}^{\ell}_{ag} = \langle S \times \mathcal{L}, A, Tr^{\ell}_{ag}, \bar{R}\rangle$ is a learning agent, $f_r : \mathcal{L}^* \to \mathbb{R}$ is a high-level reward function, and $f_c : \mathcal{L}^* \to \{0, 1\}$ is a clock function, find a policy $\bar{\rho} : (S \times \mathcal{L})^* \to A$ such that $v^{\bar{\rho}}(s_0)$ is maximized.

Observe that this setting is rather general since f_r and f_c are only assumed to be regular functions of histories. Two popular ways of representing regular functions is either via finite-state machines, like transducers, or via declarative languages (e.g. temporal logics). In the next sections, we consider both cases and provide a solution technique to solve our problem.

4 Clocked Reward Machine

In this section, we consider the case where f_r and f_c are specified as finite-state machines. In particular, we consider a *reward transducer* (or *reward machine*) $M_r = \langle Q_r, \mathcal{L}, \mathcal{R}, q^r_0, \delta_r, \theta_r\rangle$ with $\mathcal{R} \subseteq \mathbb{R}$ the output alphabet, i.e. a finite set of reward values, and a *clock transducer* (or *clock machine*) $M_c = \langle Q_c, \mathcal{L}, \{0, 1\}, q^c_0, \delta_c, \theta_c\rangle$. In particular, for a trace $t = \ell_0, \ldots, \ell_n$, we define $f_{M_r}(t) = \theta_r(\delta^*_r(q^r_0, \ell_0, \ldots, \ell_{n-1}), \ell_n)$ as the reward function of M_r, i.e. $f_{M_r}(t)$ is the last

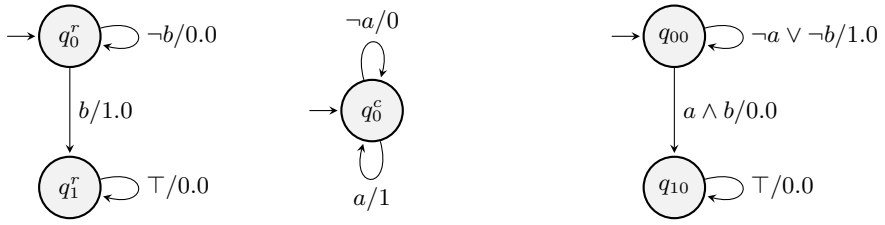


Fig. 3: M_r, M_c, and M_{cr} of Example 3.

outputted reward by M_r on input t, while $f_{M_c}(t) = \theta_c(\delta^*(q_0, \ell_0, \ldots, \ell_{n-1}), \ell_n)$ as the clock function of M_c. The reward transducer is a well-known concept in reinforcement learning for high-level task specifications, e.g. see [14, 15, 4].

We show how, given M_c and M_r, we can compute a new Mealy machine $M_{cr} = \langle Q_{cr}, \mathcal{L}, \mathcal{R} \cup \{0\}, q_0^{cr}, \delta_{cr}, \theta_{cr} \rangle$, that we call the *clocked reward machine*, such that the function it represents is precisely $\bar{R}_{cr}$ when $f_r = f_{M_r}$ and $f_c = f_{M_c}$. Such machine M_{cr} is defined as follows:

- $Q_{cr} = Q_c \times Q_r$;
- $q^{cr} = (q_0^c, q_0^r)$;
- $\delta_{cr}((q_c, q_r), \ell) = \begin{cases} (\delta_c(q_c, \ell), q_r) & \text{if } \theta_c(q_c, \ell) = 0 \\ (\delta_c(q_c, \ell), \delta_r(q_r, \ell)) & \text{if } \theta_c(q_c, \ell) = 1 \end{cases}$;
- $\theta_{cr}((q_c, q_r), \ell) = \begin{cases} 0 & \text{if } \theta_c(q_c, \ell) = 0 \\ \theta_r(q_r, \ell) & \text{if } \theta_c(q_c, \ell) = 1 \end{cases}$

Intuitively, the clocked reward machine is like the classical synchronous product between two transducers, except that the state component coming from the reward machine q_r is progressed only if the clock component q^c, after reading the symbol ℓ, is such that the clock output is 1. The *clocked reward function corresponding to* M_{cr} is $\bar{R}_{M_{cr}}(t) = \sum_{i=0}^{n} \gamma^i \theta_{cr}(\delta^*(q_0^{cr}, t[0:i-1]), t[i])$.
The correctness follows by construction:

Theorem 1. *Let M_r and M_c be a reward machine and a clock machine, respectively, and let f_r and f_c their reward and clock functions. Let the clocked reward machine M_{cr}. Moreover, let $\bar{R}_{cr}(t)$ be a clocked reward function with $f_r = f_{M_r}$ and $f_c = f_{M_c}$. We have that, for all traces t, $\bar{R}_{M_{cr}}(t) = \bar{R}_{cr}(t)$*

Proof. We prove the claim by induction on the length of the trace $t = \ell_1, \ldots, \ell_n$. If $t = \epsilon$, then $\bar{R}_{M_{cr}}(t) = \bar{R}_{cr}(t) = 0$. Now assume the claim holds for $t_{n-1} = \ell_1, \ldots, \ell_{n-1}$, and let $t_n = t_{n-1}\ell_n$, Let $q_{n-1}^{cr} = \delta_{cr}^*(t_{n-1})$ be the last state of the run over trace t_{n-1}. On one hand, we have $\bar{R}_{M_{cr}}(t_n) = \bar{R}_{M_{cr}}(t_{n-1}) + \gamma^n \theta_{cr}(q_{n-1}^{cr}, \ell_n)$, by definition of $\bar{R}_{M_{cr}}$, while on the other hand we have $\bar{R}_{cr}(t_n) = \bar{R}_{cr}(t_{n-1}) + \gamma^n f_{M_r}(t_n|_{f_{M_c}}) \cdot f_{M_c}(t_n)$, by definition of $\bar{R}_{cr}$ (Equation 1) and by assumption. Since $\bar{R}_{M_{cr}}(t_{n-1}) = \bar{R}_{cr}(t_{n-1})$ by inductive hypothesis, it remains to prove that $\theta_{cr}(q_{n-1}^{cr}, \ell_n) = f_{M_r}(t_n|_{f_{M_c}}) \cdot f_{M_c}(t_n)$. We have two cases: either $\theta_c(q_{n-1}^c, \ell_n) = 0$, or $\theta_c(q_{n-1}^c, \ell_n) = 1$. In the former case, by construction of

θ_{cr}, we have $\theta_{cr}(q^{cr}_{n-1}, \ell_n) = f_{M_c}(t_n) = 0$. Hence, the claim holds. In the latter case, on one hand we have $\theta_{cr}(q^{cr}_{n-1}, \ell_n) = \theta_r(q^r_{n-1}, \ell_n)$, by definition of θ_{cr}, and on the other hand $f_{M_r}(t_n) = \theta_r(q^r_{n-1}, \ell_n)$. Hence, both terms are equal.

Example 3. Let M_r be the reward machine for the goal "reach b" (Figure 3, left), and let $\mathcal{A}_c$ be the clock machine that ticks whenever a is true (Figure 3, middle). The clock product M_{cr} is shown in Figure 3, right. Note that the transition of the M_r-component of the state is made only if the clock is high, i.e. when the symbol a holds in the current timestep.

Based on the clocked reward machine construction, we now provide a solution to Problem 1 in case f_r and f_c are specified as Mealy machines M_r and M_c, respectively. Starting from $\mathcal{M}^{\ell}_{ag}$ and M_{cr}, we construct the MDP $\mathcal{M}' = \langle S', A', Tr', R' \rangle$, defined as follows:

- $S' = S \times \mathcal{L} \times Q_{cr}$;
- $A' = A$
- $Tr'((s, \ell, q), a, (s', \ell', q')) = \begin{cases} Tr(s, a, s') & \text{if } q' = \delta(q, \ell') \\ 0 & \text{otherwise.} \end{cases}$
- $R'((s, \ell, q), a, (s', \ell', q')) = R_{ag}(s, a, s') + \theta_{cr}(q, \ell')$

By construction, and by Theorem 1, it holds that the MDP $\mathcal{M}'$ is equivalent to the NMRDP $\mathcal{M}^{\ell}_{ag}$, in the sense of [1], and therefore optimal policies ρ' for $\mathcal{M}'$ can be transformed in optimal policies for $\mathcal{M}^{\ell}_{ag}$:

Theorem 2. *An optimal policy for the NMRDP $\mathcal{M}^{\ell}_{ag}$ with $f_r = f_{M_r}$ and $f_c = f_{M_c}$, can be learned by learning corresponding optimal policies for the MDP $\mathcal{M}'$.*

In other words, one can solve Problem 1 by first finding an optimal (memoryless) policy ρ' for $\mathcal{M}'$, and then by defining an equivalent policy on $\mathcal{M}^{\ell}_{ag}$, as follows: let $\tau = (s_0, \ell_0), a_1, (s_1, \ell_1), \ldots, (s_{n-1}, \ell_{n-1}, a_n)$ be the current trajectory of the process leading to state (s_n, ℓ_n). Let q_n denote the current state of Mealy machine M_{cr}, given input $t = \ell_0, \ldots, \ell_n$. Then, we define $\bar{\rho}(\tau) := \rho'(s_n, \ell_n, q_n)$.

In fact, by using a technique analogous to [5], one can show that we can restrict the policies of interest by dropping the fluents configurations $\mathcal{L}$ from the agent features. Hence, the resulting state space of the new MDP $\mathcal{M}''$ would be $S \times Q_{cr}$, with the state component Q_{cr} being progressed correctly by the environment.

Theorem 3. *An optimal policy for the NMRDP $\mathcal{M}^{\ell}_{ag}$ with $f_r = f_{M_r}$ and $f_c = f_{M_c}$ can be learned by learning corresponding optimal policies for the MDP $\mathcal{M}''$.*

Proof sketch. By Theorem 2, there exist an optimal policy ρ' such that $\bar{\rho}$ computed from ρ is also optimal for $\mathcal{M}^{\ell}_{ag}$. There exists a corresponding optimal policy for $\mathcal{M}''$, $\rho'' : S \times Q_{cr} \to A$, which differs from ρ' by dropping the $\mathcal{L}$ component of the state; optimality of ρ'' wrt $\mathcal{M}''$ can be shown by marginalizing the transition function distribution T'' over ℓ' (see proof of Theorem 6 in [5]).

As a consequence of Theorem 3, one can solve the learning problem of Problem 1 by learning an optimal policy for $\mathcal{M}''$.

5 Declarative Clock Specifications

In this section, we study a variant of Problem 1 where the reward and clock functions are specified *declaratively* using a formal regular language, e.g. LTL_f and LDL_f. The main advantages of doing so are *(i)* the use of a high-level, human-understandable language, *(ii)* succinctness with respect to the finite-state machine formalism (in the best case, a doubly-exponential gain), and *(iii)* better modularity and composability of specifications. Note that any logic formalism with finite trace-based semantics that is not more expressive than regular expressions can be used in our framework (e.g. Pure-Past LTL [7]). At a high level, the solution method in the declarative setting works by transforming both the reward and clock specifications into a reward machine and a clock machine; then, we rely on the solution introduced in Section 4.

More formally, we have a *reward specification* (φ_r, r), where φ_r is the $\text{LTL}_f/\text{LDL}_f$ formula that has to be satisfied in order to give the reward signal r to the agent (as in [2, 5, 4]), plus a *clock specification* φ_c, another $\text{LTL}_f/\text{LDL}_f$ formula which specifies the clock function. Their respective reward function f_{φ_r} and the clock function f_{φ_c} are defined as follows:

$$f_{\varphi_r}(t) = \begin{cases} r & \text{if } t \models \varphi_r \\ 0 & \text{otherwise.} \end{cases} \qquad f_{\varphi_c} = \begin{cases} 1 & \text{if } t \models \varphi_c \\ 0 & \text{otherwise.} \end{cases}$$

Intuitively, the reward function $f_{\varphi_r}(t)$ returns r whenever $t \models \varphi_r$, otherwise it gives no reward signal. Similarly, the clock function $f_{\varphi_c}(t)$ is 1 iff $t \models \varphi_c$. A *clocked reward specification* is the triple $(\varphi_r, \varphi_c, r)$, and the derived clocked reward function $\bar{R}_{\varphi_r,\varphi_c}$ is defined by starting from $\bar{R}_{cr}$ and by setting $f_r = f_{\varphi_r}$ and $f_c = f_{\varphi_c}$.

Example 4. Continuing Example 3, let the reward specification be (φ_r, r), where $\varphi_r = \Diamond b$ and $r = 1$, and $\varphi_c = \Diamond(a \wedge \mathit{last})$. We have that $f_{\varphi_r} = f_{M_r}$ and $f_{\varphi_c} = f_{M_c}$.

In order to solve this declarative variant of Problem 1, we resort to a reduction to the solution shown in Section 4. To do so, we proceed in steps. First, from the reward specification (φ_r, r), we compute the DFA equivalent to φ_r, $\mathcal{A}_{\varphi_r} = \langle Q, \mathcal{L}, q_0, F, \delta \rangle$. Then, we define the Moore machine $M' = \langle Q, \mathcal{L}, \{0, r\}, q_0, \delta, \theta \rangle$ where $\theta(q) = r$ if $q \in F$, otherwise $\theta(q) = 0$. From M', we can compute its equivalent Mealy reward machine M_{φ_r}. An analogous transformation can be made for the clock specification, i.e. from φ_c to the clock machine M_{φ_c}. Finally, we can resort to the solution presented in Section 4 to solve our problem.

Theorem 4. *Let $\langle \mathcal{M}^{\ell}_{ag}, f_{\varphi_r}, f_{\varphi_c} \rangle$ be an instance of Problem 1 in which f_{φ_r} and f_{φ_c} are specified by a $\text{LTL}_f/\text{LDL}_f$ reward specification (φ_r, r) and φ_c, respectively. Then, optimal policies for $\langle M^{\ell}_{ag}, f_{M_{\varphi_r}}, f_{M_{\varphi_c}} \rangle$ are also optimal policies for $\langle \mathcal{M}^{\ell}_{ag}, f_{\varphi_r}, f_{\varphi_c} \rangle$.*

Proof. By construction and by Theorem 1.

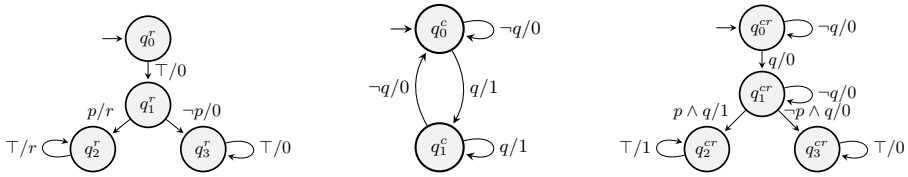


Fig. 4: Machines for Example 5, from left to right: reward machine for $(\bigcirc p, r)$, clock machine for φ_c, and reward machine for the unlocked-equivalent specification (φ_{cr}, r)

6 Unclocked-Equivalent Specifications

Given a $\text{LTL}_f/\text{LDL}_f$ clocked reward specification $(\varphi_r, \varphi_c, r)$ we ask ourselves whether there exist an *unclocked-equivalent reward specification* (φ_{cr}, r) such that for all traces t, $\bar{R}_{\varphi_{cr}}(t) = \bar{R}_{cr}(t)$, where $\bar{R}_{\varphi}(t) = \sum_{i=0:t[0:i]\models\varphi}^{n} \gamma^i r$. We answer positively, and we explain a way to compute φ_{cr}, given a clocked specification $(\varphi_r, \varphi_c, r)$. The following theorem shows how, by construction:

Theorem 5. *Given a clocked reward specification $(\varphi_r, \varphi_c, r)$, there exist a LDL_f reward specification (φ_{cr}, r) such that $\bar{R}_{\varphi_r,\varphi_c} = \bar{R}_{cr}$.*

Proof. First, we compute the DFAs $\mathcal{A}_r$ and $\mathcal{A}_c$, which are the DFAs equivalent to φ_r and φ_c, respectively. Then, we consider their equivalent Mealy machines $M_{\mathcal{A}_r}$ and $M_{\mathcal{A}_c}$, as explained in the Preliminaries, from which we can compute M_{cr} using the clocked product (see Section 4, with the only difference that the output alphabet is not a set of rewards f_r but instead $\{accept\}$). Since M_c behaves as an acceptor, we can compute its equivalent DFA $\mathcal{A}_{cr}$. Then, we can compute an equivalent regular expression for $\mathcal{A}_{cr}$ in exponential time, and thus get a regular expression that is at most exponentially-larger than the DFA [13]. Finally, we can convert the regular expression to an LDL_f formula φ_{cr} with constant blow-up [8]. By construction, $t \models \varphi$ iff M_{cr} with input t would have outputted *accept* in place of r, iff the clock condition was satisfied (Theorem 1).

In fact, if φ_r and φ_c are both LTL_f formulas, then the language recognized by the *clocked* DFA *product* $\mathcal{A}_{cr}$ (i.e. the acceptor version of M_{cr}) is a star-free language, and so it can be defined by some LTL_f formula [8].

Theorem 6. *If $(\varphi_r, \varphi_c, r)$ is a LTL_f clocked specification, then there exist an unclocked-equivalent LTL_f reward specification (φ_{cr}, r).*

Proof. The crux of the proof is to show that the clocked product $\mathcal{A}_{cr}$ is a counter-free automaton [19], and therefore $\mathcal{L}(\mathcal{A}_{cr})$ is a star-free language [25]. The claim follows since the class of star-free regular languages is equivalent to the class of LTL_f-definable languages [8], and by the equivalence of $\mathcal{A}_{cr}$ with the clocked semantics for $(\varphi_r, \varphi_c, r)$ as per Theorem 1. First, observe that both $\mathcal{A}_r$ and $\mathcal{A}_c$ are counter-free automata, since they are semantically equivalent to LTL_f formulas

φ_r and φ_c, respectively. Assume by contradiction that $\mathcal{A}_{cr}$ has a *permutation*, i.e. for some set $P = \{q_1, \ldots, q_m\}$, $m \geq 2$, of states of $\mathcal{A}_{cr}$, there is a run $q_1 \ldots q_m$ such that $\delta'(q_i, \ell_i) = q_{i+1}$, for $1 \leq i \leq m-1$, and $\delta'(q_m, \ell_m) = q_1$. Now, consider the same set of states but only considering the state components coming from $\mathcal{A}_c$, i.e. $\{q_1^c, \ldots, q_m^c\}$. Note that there cannot be self-loops in this path, i.e. $q_i^c \neq q_j^c$ for all $i \neq j$. By construction of clocked product, and in particular by definition of δ', it is easy to see that P is also a permutation for $\mathcal{A}_c$. Since for a DFA being permutation-free is equivalent to being counter-free [19], we have that $\mathcal{A}_c$ does have a counter, and therefore we get a contradiction.

Differently from Theorem 5, Theorem 6 only tells us that an unclocked-equivalent LTL_f exists, but not how to compute it. Now we give a possible automata-based approach for the LTL_f case. Given a clocked LTL_f specification, compute $\mathcal{A}_{cr}$, which can be doubly-exponentially larger. Then, reverse all transitions to get an NFA $\mathcal{A}^R$ that accepts the reverse of the language of $\mathcal{A}_{cr}$, then determinize this NFA to get an equivalent DFA $\mathcal{A}'^R$. Note that $\mathcal{A}'^R$ may be exponentially larger than $\mathcal{A}^R$. Now, apply Theorem 11 of [7] to transform this DFA into an equivalent PLTL_f formula ψ. Finally, form the swap ψ^{sw} for the reverse language of ψ. Then, ψ^{sw} is the LTL_f formula equivalent to the PLTL_f formula φ.

Such translations for LDL_f (resp. LTL_f) are very impractical, as we incur in three (resp. four) exponential blowups in the size of the original clocked specification. It would be interesting to devise direct translations from clocked $\text{LTL}_f/\text{LDL}_f$ specifications into classical $\text{LTL}_f/\text{LDL}_f$ formulas, but we leave this as future work.

Example 5. In this example, we give an idea about how an unclocked-equivalent formula can be more verbose and counterintuitive than a clocked specification. Consider $(\varphi_r, \varphi_c, r)$, with $\varphi_r = \bigcirc p$ and $\varphi_c = \Diamond(q \wedge \mathit{last})$. The clock formula intuitively means "evaluate the goal formula only when q is true in the current timestep". The equivalent machines for these specifications are shown in Figure 4. Intuitively, the clocked specification transitions to the next state only when q holds. However, the second time this happens, p must hold to satisfy the specification. It can be shown that the formula $\varphi_{cr} = \neg q \,\mathcal{U}(q \wedge \bigcirc(\neg q \,\mathcal{U}(p \wedge q)))$ is unclocked-equivalent to the clocked reward specification. The formula includes an $\mathcal{U}$ operator and a nested $\bigcirc$ and $\mathcal{U}$ operator.

7 Conclusion

In this paper, we investigated the separation of a clock specification from the temporal specification itself, allowing the time granularity of the temporal specification to be coarser than the actual time. Our work can be extended to consider multi-clocked specifications, which are needed for certain applications. One nice example is reported in [12] where temporal specifications for space missions are formalized in a variant of LTL_f at different time-granularities, called *types*, to represents conditions with different frequencies like second, hours, days, etc. In this context, our work may give the basis for developing sophisticated multi-clocked specifications where clocks are specified in $\text{LTL}_f/\text{LDL}_f$.

Acknowledgements

This work has been partially supported by the ERC-ADG WhiteMech (No. 834228), the PRIN project RIPER (No. 20203FFYLK), the PNRR MUR project FAIR (No. PE0000013), and the Sapienza project MARLeN (Multi-layer Abstraction for Reinforcement Learning with Non-Markovian Rewards).

References

1. Bacchus, F., Boutilier, C., Grove, A.J.: Rewarding behaviors. In: AAAI/IAAI, Vol. 2. pp. 1160–1167. AAAI Press / The MIT Press (1996)
2. Brafman, R.I., De Giacomo, G., Patrizi, F.: Ltlf/ldlf non-markovian rewards. In: AAAI. pp. 1771–1778. AAAI Press (2018)
3. De Giacomo, G., Favorito, M., Iocchi, L., Patrizi, F.: Imitation learning over heterogeneous agents with restraining bolts. In: ICAPS. pp. 517–521. AAAI Press (2020)
4. De Giacomo, G., Favorito, M., Iocchi, L., Patrizi, F., Ronca, A.: Temporal logic monitoring rewards via transducers. In: KR. pp. 860–870 (2020)
5. De Giacomo, G., Iocchi, L., Favorito, M., Patrizi, F.: Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications. In: ICAPS. pp. 128–136. AAAI Press (2019)
6. De Giacomo, G., Iocchi, L., Favorito, M., Patrizi, F.: Restraining bolts for reinforcement learning agents. In: AAAI. pp. 13659–13662. AAAI Press (2020)
7. De Giacomo, G., Stasio, A.D., Fuggitti, F., Rubin, S.: Pure-past linear temporal and dynamic logic on finite traces. In: IJCAI. pp. 4959–4965. ijcai.org (2020)
8. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: IJCAI. pp. 854–860. IJCAI/AAAI (2013)
9. Eisner, C., Fisman, D., Havlicek, J., McIsaac, A., Campenhout, D.V.: The definition of a temporal clock operator. In: ICALP. Lecture Notes in Computer Science, vol. 2719, pp. 857–870. Springer (2003)
10. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. J. Comput. Syst. Sci. **18** (1979)
11. Harel, D.: Dynamic logic. In: Handbook of philosophical logic, pp. 497–604. Springer (1984)
12. Hariharan, G., Kempa, B., Wongpiromsarn, T., Jones, P.H., Rozier, K.Y.: MLTL multi-type (MLTLM): A logic for reasoning about signals of different types. In: NSV/FoMLAS@CAV. Lecture Notes in Computer Science, vol. 13466, pp. 187–204. Springer (2022)
13. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)
14. Icarte, R.T., Klassen, T.Q., Valenzano, R.A., McIlraith, S.A.: Using reward machines for high-level task specification and decomposition in reinforcement learning. In: ICML. Proceedings of Machine Learning Research, vol. 80, pp. 2112–2121. PMLR (2018)
15. Icarte, R.T., Klassen, T.Q., Valenzano, R.A., McIlraith, S.A.: Reward machines: Exploiting reward function structure in reinforcement learning. J. Artif. Intell. Res. **73**, 173–208 (2022). https://doi.org/10.1613/JAIR.1.12440, https://doi.org/10.1613/jair.1.12440

16. Li, X., Vasile, C.I., Belta, C.: Reinforcement learning with temporal logic rewards. In: IROS. pp. 3834–3839. IEEE (2017)
17. Linz, P., Rodger, S.H.: An introduction to formal languages and automata. Jones & Bartlett Learning (2022)
18. Littman, M.L., Topcu, U., Fu, J., Jr., C.L.I., Wen, M., MacGlashan, J.: Environment-independent task specifications via GLTL. CoRR **abs/1704.04341** (2017)
19. McNaughton, R., Papert, S.A.: Counter-Free Automata (MIT research monograph no. 65). The MIT Press (1971)
20. Mealy, G.H.: A method for synthesizing sequential circuits. The Bell System Technical Journal **34**(5), 1045–1079 (1955)
21. Moore, E.F.: Gedanken-experiments on sequential machines. In: Automata Studies.(AM-34), Volume 34, pp. 129–154. Princeton University Press (2016)
22. Pnueli, A.: The temporal logic of programs. In: FOCS. pp. 46–57. IEEE Computer Society (1977)
23. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley Series in Probability and Statistics, Wiley (1994)
24. Rabin, M.O., Scott, D.S.: Finite automata and their decision problems. IBM J. Res. Dev. **3**(2), 114–125 (1959)
25. Schützenberger, M.P.: On finite monoids having only trivial subgroups. Inf. Control. **8**(2), 190–194 (1965)
26. Sutton, R.S., Barto, A.G.: Reinforcement learning - an introduction. Adaptive computation and machine learning, MIT Press (1998)
27. Vardi, M.Y.: The rise and fall of linear time logic. In: GandALF (2011), http://www.cs.rice.edu/ vardi/papers/gandalf11-myv.pdf
28. Xu, Z., Topcu, U.: Transfer of temporal logic formulas in reinforcement learning. In: IJCAI. pp. 4010–4018. ijcai.org (2019)