Specifying Non-Markovian Rewards in MDPs Using LDL on Finite Traces

#3008

Abstract

In Markov Decision Processes (MDPs), the reward obtained in a state depends on the properties of the last state and action. This state dependency makes it difficult to reward more interesting long-term behaviors, such as always closing a door after it has been opened, or providing coffee only following a request. Extending MDPs to handle such non-Markovian reward function was the subject of two previous lines of work, both using variants of LTL to specify the reward function and then compiling the new model back into a Markovian model. Building upon recent progress in the theories of temporal logics over finite traces, we adopt LDL_f for specifying non-Markovian rewards and provide an elegant automata construction for building a Markovian model, which extends that of previous work and offers strong minimality and compositionality guarantees.

1 Introduction

Markov Decision Processes (MDPs) are a central model for sequential decision making under uncertainty. They are used to model and solve many real-world problems, and to address the problem of learning to behave well in unknown environments. The Markov assumption is a key element of this model. It states that the effects of an action depend only on the state in which it was executed, and that reward given at a state depends only on the previous action and state. It has long been observed [Bacchus *et al.*, 1996; Thiébaux *et al.*, 2006] that many performance criteria call for more sophisticated reward functions that do not depend on the last state only. For example, we may want to reward a robot that *eventually delivers coffee each time it gets a request;* or, to reward the robot for keeping its operating environment tidy, e.g., by *closing doors after opening*; or, to ensure it will *access restricted areas only after having acquired the right permission*. All these rewards are non-Markovian. Recently, Littman, in his IJCAI 2015 invited talk "Programming Agents via Rewards," advocated that it may actually be more convenient, from a design perspective, to assign rewards to the satisfaction of declarative temporal properties, rather than to states.

To extend MDPs with non-Markovian rewards we need a language for specifying such rewards. Markovian rewards are specified as a function R from the previous state and action to the reals. R can be specified using an explicit reward matrix, or implicitly, by associating a reward with properties of the last state and action. With non-Markovian rewards, an explicit representation is no longer possible, as the number of possible histories or futures of a state is infinite. Hence, we must use an implicit specification that can express properties of past (or future) states. To date, two specification languages have been proposed. [Bacchus *et al.*, 1996] suggests using a temporal logic of the past. Whether state s_i satisfies such a past temporal formula depends on the entire sequence of states leading up to it: $s_1, s_2, \ldots, s_i$. Thus, we can reward appropriate response to a "bring-coffee" command by associating a reward with the property *the "bring-coffee" command was issued in the past, and now I have coffee*. A second proposal, by [Thiébaux *et al.*, 2006], uses a temporal logic of the future with a special symbol to denote awarding a reward. At each step, one checks whether this symbol must be true in the current state, for the reward formula to be satisfied in the initial state. If that is the case, the current state is rewarded. This language is a little less intuitive, and its semantics is more complicated.

Existing MDP solution methods, possibly with the exception of Monte-Carlo tree search algorithms, rely heavily on the Markov assumption, and cannot be applied directly with non-Markovian rewards. To address this, both proposals above transform the non-Markovian model to an equivalent Markovian one that can be solved using existing algorithms, by enriching the state with information about the past. For example, suppose we extend our state to record whether a "bring-coffee" command was issued earlier. A reward for bringing coffee in states that indicate that the "bring-coffee" command was issued in the past, is now Markovian. It rewards the same behaviors as the earlier non-Markovian reward on the original state. We call a model obtained by extending the state space of the original non-Markovian MDP, an *extended MDP*.

Using an extended MDP is a well-known idea. Since state space size affects the practical and theoretical complexity of most MDP solution algorithms, the main question is how to *minimally* enrich the state so as to make rewards Markovian. [Bacchus *et al.*, 1996] provide algorithms for constructing an extended MDP that attempt to minimize size by reducing the amount of information about the past that is maintained. While their construction does not generate the minimal extended MDP, they allude to using automata minimization techniques to accomplish this. [Thiébaux *et al.*, 2006], instead, use a construction that works well with forward search based algorithms, such as LAO* [Hansen and Zilberstein, 2001] and LRTDP [Bonet and Geffner, 2003]. Unlike classical dynamic programming methods that require the entire state space a-priori, these algorithms generate reachable states only. With a good heuristic function, they often generate only a fraction of the state space. So, while the augmented search space they obtain is not minimal, because states are constructed on the fly during forward search, their approach does not require a-priori enumeration of the state space, and never generates an unreachable state. They call this property *blind minimality*.

The aim of this paper is to bring to bear developments in the theory of temporal logic over finite traces to the problem of specifying and solving MDP with non-Markovian rewards. We adopt LDL_f, a temporal logic of the future interpreted over finite traces, which extends LTL_f, the classical linear-time temporal logic over finite traces [De Giacomo and Vardi, 2013]. LDL_f has the same computational features of LTL_f but it is more expressive, as it captures monadic second order logic (MSO) on finite traces (i.e., inductively defined properties), instead of first-order logic (FO), as LTL_f. A number of techniques based on automata manipulation have been developed for LDL_f, to address tasks such as satisfiability, model checking, reactive synthesis, and planning under full/partial observability [De Giacomo and Vardi, 2013; De Giacomo and Vardi, 2015; De Giacomo and Vardi, 2016; Torres and Baier, 2015; Camacho *et al.*, 2017]. We exploit such techniques to generate an extended MDP with good properties.

Our formalism has three important advantages: 1. **Enhanced expressive power.** We move from linear-time temporal logics used by past authors to LDL_f, paying no additional (worst-case) complexity costs. LDL_f can encode in polynomial time LTL_f, regular expressions (RE), the past LTL (PLTL) of [Bacchus *et al.*, 1996], and virtually all examples of [Thiébaux *et al.*, 2006]. Often, LDL_f can represent more compactly and more intuitively conditions specified in LTL_f or PLTL. Future logics are more commonly used in the model checking community, as they are considered more natural for expressing desirable properties. This is especially true with complex properties that require the power of LDL_f. 2. **Minimality and Compositionality.** We generate a minimal equivalent extended MDP, exploiting existing techniques for constructing automata that track the satisfiability of an LDL_f formula. This construction is relatively simple and compositional: if a new reward formula is to be added, we only need to optimize the corresponding automaton and add it to the current (extended) MDP. If the current MDP was minimal, the resulting (extended) MDP is minimal, too. 3. **Blind Minimality.** The automaton used to identify when a reward should be given can be constructed in a forward manner using progression, providing blind minimality as in [Thiébaux *et al.*, 2006]. If, instead, we want pure minimality, unlike the construction of [Bacchus *et al.*, 1996], we can exploit the forward construction to never generate unreachable states, before applying automata minimization. Hence, we have the best of both worlds.

2 Background

MDPs. A Markov Decision Process (MDP) $\mathcal{M} = \langle S, A, Tr, R\rangle$ consists of 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. We can further restrict actions to be defined on a subset of S only, and use $A(s)$ to denote the actions applicable in s. The reward function, $R : S \times A \to \mathbb{R}$, specifies the real-valued reward received by the agent when applying action a in state s. In this paper, states in S are truth assignments to a set $\mathcal{P}$ of primitive propositions. Hence, if φ is a propositional formula and s a state, we can check whether $s \models \varphi$.

A solution to an MDP, called a *policy*, assigns an action to each state, possibly conditioned on past states and actions. The *value* of policy ρ at s, $v^\rho(s)$, is the expected sum of (possibly discounted) rewards when starting at state s and selecting actions based on ρ. Every MDP has an *optimal* policy, ρ^*, i.e., one that maximizes the expected sum of rewards for every starting state $s \in S$. In the case of infinite horizon, there exists an optimal policy that is stationary $\rho : S \to A$ (i.e., ρ depends only on the current state) and deterministic [Puterman, 2005]. There are diverse methods for computing an optimal policy. With the exception of online simulation-based methods, they rely on the fact that both transitions and rewards depend on the last state only – i.e., are independent of earlier states and actions (*Markov property*). The theoretical and practical complexity of solution algorithms is strongly impacted by $|S|$.

LTL_f and LDL_f. LTL_f is essentially LTL [Pnueli, 1977] interpreted over finite, instead of infinite, traces. LTL_f is as expressive as FO over finite traces and star-free RE, thus strictly less expressive than RE, which in turn are as expressive as MSO over finite traces. RE themselves are not convenient for expressing temporal specifications, since, e.g., they miss direct constructs for negation and conjunction. For this reason, [De Giacomo and Vardi, 2013] introduced LDL_f (*linear dynamic logic on finite traces*), which merges LTL_f with RE, through the syntax of the well-known logic of programs PDL, *propositional dynamic logic* [Fischer and Ladner, 1979; Harel *et al.*, 2000; Vardi, 2011], but interpreted over finite traces.

We consider a variant of LDL_f that works also on empty traces. Formally, LDL_f formulas φ are built as follows:

$$\begin{array}{lll}\varphi & ::= & tt \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle\rho\rangle\varphi \\ \rho & ::= & \phi \mid \varphi? \mid \rho_1 + \rho_2 \mid \rho_1;\rho_2 \mid \rho^*\end{array}$$

where tt stands for logical true; ϕ is a propositional formula over $\mathcal{P}$ (including $true$, not to be confused with tt); ρ denotes path expressions, which are RE over propositional formulas ϕ with the addition of the test construct $\varphi?$ typical of PDL. We use abbreviations $[\rho]\varphi \doteq \neg\langle\rho\rangle\neg\varphi$, as in PDL, $ff \doteq \neg tt$, to denote false, and $\phi \doteq \langle\phi\rangle tt$ to denote the occurence of proposition ϕ.

Intuitively, $\langle\rho\rangle\varphi$ 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 $[\rho]\varphi$ 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.

The semantics of LDL_f is given in terms of *finite traces*, i.e., finite sequences $\pi = \pi_0, \ldots, \pi_n$ of elements from the alphabet $2^{\mathcal{P}}$. We define $\pi(i) \doteq \pi_i$, $length(\pi) \doteq n+1$, and $\pi(i,j) \doteq \pi_i, \pi_{i+1}, \dots, \pi_j$. When $j > n$, $\pi(i,j) \doteq \pi(i,n)$.

In decision processes, traces are usually sequences of states *and* actions, i.e., they have the form: $\langle s_0, a_1, s_1, \dots, s_{n-1}, a_n \rangle$. These can still be represented as traces of the form $\pi = \pi_0, \dots, \pi_n$, by extending the set $\mathcal{P}$ to include one proposition p_a per action a, and setting $\pi_i \doteq s_i \cup \{p_a \mid a = a_{i+1}\}$. In this way, π_i denotes the pair (s_i, a_{i+1}). We will always assume this form, even if referring to sequences of states and actions. Given a finite trace π, an LDL_f formula φ, and a position i, we define when φ *is true* at step i, written $\pi, i \models \varphi$, by (mutual) induction, as follows:

- $\pi, i \models tt$;
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$;
- $\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$;
- $\pi, i \models \langle\rho\rangle\varphi$ iff there exists $i \leq j \leq length(\pi)$ such that $\pi(i,j) \in \mathcal{L}(\rho)$ and $\pi, j \models \varphi$, where the relation $\pi(i,j) \in \mathcal{L}(\rho)$ is as follows:
  - $\pi(i,j) \in \mathcal{L}(\phi)$ if $j{=}i{+}1$, $j \leq length(\pi)$, and $\pi(i) \models \phi$ (ϕ propositional);
  - $\pi(i,j) \in \mathcal{L}(\varphi?)$ if $j = i$ and $\pi, i \models \varphi$;
  - $\pi(i,j) \in \mathcal{L}(\rho_1 + \rho_2)$ if $\pi(i,j) \in \mathcal{L}(\rho_1)$ or $\pi(i,j) \in \mathcal{L}(\rho_2)$;
  - $\pi(i,j) \in \mathcal{L}(\rho_1; \rho_2)$ if there exists k such that $\pi(i,k) \in \mathcal{L}(\rho_1)$ and $\pi(k,j) \in \mathcal{L}(\rho_2)$;
  - $\pi(i,j) \in \mathcal{L}(\rho^*)$ if $j = i$ or there exists k such that $\pi(i,k) \in \mathcal{L}(\rho)$ and $\pi(k,j) \in \mathcal{L}(\rho^*)$.

Observe that if $i > length(\pi)$, the above definitions still apply. In particular, $[\rho]\varphi$ is trivially true and $\langle\rho\rangle\varphi$ trivially false.

We say that a trace π *satisfies* an LDL_f formula φ, written $\pi \models \varphi$, if $\pi, 0 \models \varphi$. Also, sometimes we denote by $\mathcal{L}(\varphi)$ the set of traces that satisfy φ: $\mathcal{L}(\varphi) = \{\pi \mid \pi \models \varphi\}$.

LDL_f is as expressive as MSO over finite words. It captures LTL_f, by seeing *next* and *until* as the abbreviations $\circ\varphi \doteq \langle true\rangle\varphi$ and $\varphi_1 \,\mathcal{U}\, \varphi_2 \doteq \langle(\varphi_1?; true)^*\rangle\varphi_2$, and any RE r, with the formula $\langle r\rangle end$, where $end \doteq [true?]f\!f$ expresses that the trace has ended. Note that in addition to end we can also denote the last element of the trace as $last \doteq [true]end$ or equivalently $last \doteq [true]f\!f$. The latter has also an LTL_f-equivalent: $\neg\circ true$, instead end does not.

The properties mentioned at the beginning of the introduction can be expressed in LDL_f as follows: $[true^*](request_p \rightarrow \langle true^*\rangle coffee_p)$ (all coffee requests from person p will eventually be served); $[true^*]([openDoor_d]closeDoor_d)$ (every time the robot opens door d it closes it immediately after);

$$\langle((\neg restrd_a)^*; permission_a; (\neg restrd_a)^*; restrd_a)^*; \\ (\neg restrd_a)^*\rangle end$$

(before entering restricted area a the robot must have permission for a). While the LTL_f-equivalents of the first two formulas are immediate, i.e., $\Box(request_p \rightarrow \Diamond coffee_p)$ and $\Box(openDoor_d \rightarrow \circ closeDoor_d)$, that for the third one is not.

We can associate each LDL_f formula φ with an (exponentially large) NFA A_φ that accepts exactly the traces satisfying φ. A simple direct algorithm (LDL_f2NFA) for computing the NFA given the LDL_f formula is reported below. Its correctness relies on the fact that (*i*) every LDL_f formula φ can be associated with a polynomial *alternating automaton on words* (AFW) $\mathcal{A}_\varphi$ accepting exactly the traces that satisfy φ [De Giacomo and Vardi, 2013], and (*ii*) every AFW can be transformed into an NFA, see, e.g., [De Giacomo and Vardi, 2013].

The algorithm assumes that the LDL_f formula is in *negation normal form* (NNF), i.e., with negation symbols occurring only in front of propositions (any LDL_f formula can be rewritten in NNF in linear time), and that $\mathcal{P}$ includes the special proposition $last$, denoting the last element of the trace. Let δ be the following auxiliary function, which takes in input an LDL_f formula ψ (in NNF) and a propositional interpretation Π for $\mathcal{P}$ (including $last$), and returns a positive boolean formula whose atoms are *quoted* ψ subformulas:

$$\begin{array}{rcl}
\delta(tt, \Pi) & = & true \\
\delta(f\!f, \Pi) & = & false \\
\delta(\phi, \Pi) & = & \begin{cases} true \text{ if } \Pi \models \phi \\ false \text{ if } \Pi \not\models \phi \end{cases} \quad (\phi \text{ prop.}) \\
\delta(\varphi_1 \wedge \varphi_2, \Pi) & = & \delta(\varphi_1, \Pi) \wedge \delta(\varphi_2, \Pi) \\
\delta(\varphi_1 \vee \varphi_2, \Pi) & = & \delta(\varphi_1, \Pi) \vee \delta(\varphi_2, \Pi) \\
\delta(\langle\phi\rangle\varphi, \Pi) & = & \begin{cases} \boldsymbol{E}(\varphi) \text{ if } last \notin \Pi \text{ and } \Pi \models \phi \quad (\phi \text{ prop.}) \\ \delta(\boldsymbol{E}(\varphi), \epsilon) \text{ if } last \in \Pi \text{ and } \Pi \models \phi \\ false \text{ if } \Pi \not\models \phi \end{cases} \\
\delta(\langle\psi?\rangle\varphi, \Pi) & = & \delta(\psi, \Pi) \wedge \delta(\varphi, \Pi) \\
\delta(\langle\rho_1 + \rho_2\rangle\varphi, \Pi) & = & \delta(\langle\rho_1\rangle\varphi, \Pi) \vee \delta(\langle\rho_2\rangle\varphi, \Pi) \\
\delta(\langle\rho_1; \rho_2\rangle\varphi, \Pi) & = & \delta(\langle\rho_1\rangle\langle\rho_2\rangle\varphi, \Pi) \\
\delta(\langle\rho^*\rangle\varphi, \Pi) & = & \delta(\varphi, \Pi) \vee \delta(\langle\rho\rangle\boldsymbol{F}_{\langle\rho^*\rangle\varphi}, \Pi) \\
\delta([\phi]\varphi, \Pi) & = & \begin{cases} \varphi \text{ if } last \notin \Pi \text{ and } \Pi \models \phi \quad (\phi \text{ prop.}) \\ \delta(\varphi, \epsilon) \text{ if } last \in \Pi \text{ and } \Pi \models \phi \\ true \text{ if } \Pi \not\models \phi \end{cases} \\
\delta([\psi?]\varphi, \Pi) & = & \delta(nnf(\neg\psi), \Pi) \vee \delta(\varphi, \Pi) \\
\delta([\rho_1 + \rho_2]\varphi, \Pi) & = & \delta([\rho_1]\varphi, \Pi) \wedge \delta([\rho_2]\varphi, \Pi) \\
\delta([\rho_1; \rho_2]\varphi, \Pi) & = & \delta([\rho_1][\rho_2]\varphi, \Pi) \\
\delta([\rho^*]\varphi, \Pi) & = & \delta(\varphi, \Pi) \wedge \delta([\rho]\boldsymbol{T}_{[\rho^*]\varphi}, \Pi) \\
\delta(\boldsymbol{F}_\psi, \Pi) & = & false \\
\delta(\boldsymbol{T}_\psi, \Pi) & = & true
\end{array}$$

where $\boldsymbol{E}(\varphi)$ recursively replaces in φ all occurrences of atoms of the form $\boldsymbol{T}_\psi$ and $\boldsymbol{F}_\psi$ by $\boldsymbol{E}(\psi)$; and $\delta(tt, \epsilon)$ is defined as:

$$\begin{array}{rcl}
\delta(tt, \epsilon) & = & true \\
\delta(f\!f, \epsilon) & = & false \\
\delta(\phi, \epsilon) & = & false \quad (\phi \text{ prop.}) \\
\delta(\langle\rho\varphi\rangle, \epsilon) & = & false \\
\delta([\rho]\varphi, \epsilon) & = & true
\end{array}
\qquad
\begin{array}{rcl}
\delta(\varphi_1 \wedge \varphi_2, \epsilon) & = & \delta(\varphi_1, \epsilon) \wedge \delta(\text{"}\varphi_2\text{"}, \epsilon) \\
\delta(\varphi_1 \vee \varphi_2, \epsilon) & = & \delta(\varphi_1, \epsilon) \vee \delta(\varphi_2, \epsilon) \\
\delta(\boldsymbol{F}_\varphi, \epsilon) & = & false \quad (\text{for any } \varphi) \\
\delta(\boldsymbol{T}_\varphi, \epsilon) & = & true \quad (\text{for any } \varphi)
\end{array}$$

The NFA $\mathcal{A}_\varphi$ for an LDL_f formula φ is then built in a forward fashion as shown below, where: states of $\mathcal{A}_\varphi$ are sets of atoms (recall that each atom is a quoted φ subformula) to be interpreted as conjunctions; the empty conjunction $\varnothing$ stands for $true$; q' is a set of quoted subformulas of φ denoting a minimal interpretation such that $q' \models \bigwedge_{(\psi \in q)} \delta(\psi, \Pi)$ (notice that we trivially have $(\varnothing, a, \varnothing) \in \varrho$ for every $a \in \Sigma$).

algorithm LDL_f2NFA()
input LDL_f formula φ
output NFA $\mathcal{A}_\varphi = (2^{\mathcal{P}}, \mathcal{S}, \{s_0\}, \varrho, \{s_f\})$
$s_0 \leftarrow \{\varphi\}$ ▷ single initial state
$s_f \leftarrow \varnothing$ ▷ single final state
$\mathcal{S} \leftarrow \{s_0, s_f\}$, $\varrho \leftarrow \varnothing$
while ($\mathcal{S}$ or ϱ change) **do**
  if ($q \in \mathcal{S}$ and $q' \models \bigwedge_{(\psi \in q)} \delta(\psi, \Pi)$) **then**
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{q'\}$ ▷ update set of states
    $\varrho \leftarrow \varrho \cup \{(q, \Pi, q')\}$ ▷ update transition relation

Theorem 1. *[De Giacomo and Vardi, 2015] Algorithm* LDL_f2NFA *terminates in at most an exponential number of steps, and generates a set of states $\mathcal{S}$ whose size is at most exponential in the size of the formula φ.*

Note that one can remove the proposition $last \in \mathcal{P}$ by suitably adding an extra final state to $\mathcal{A}_\varphi$ [De Giacomo and Vardi, 2015]. The NFA $\mathcal{A}_\varphi$ is correct, that is:

Theorem 2. *[De Giacomo and Vardi, 2015] For every finite trace π: $\pi \models \varphi$ iff $\pi \in L(\mathcal{A}_\varphi)$.*

Finally, we can transform the NFA $\mathcal{A}_\varphi$ into a DFA in exponential time, following the standard procedure, and then possibly put it in (the unique) minimal form, in polynomial time [Rabin and Scott, 1959]. Thus, we can transform any LDL$_f$ formula into a DFA of double exponential size. While this is a worst-case complexity, in most cases the size of the DFA is actually manageable [Tabakov and Vardi, 2005].

3 Specifying Decision Processes with Non-Markovian Rewards

In this section we extend MDPs with LDL$_f$-based reward functions resulting in a non-Markovian reward decision process (NMRDP). Then, we show how to construct an equivalent extended MDP with Markovian rewards.

A non-Markovian reward decision process (NMRDP) is a tuple $\mathcal{M} = \langle S, A, Tr, R\rangle$, where S, A and Tr are as in an MDP, and R is redefined as $R : (S \times A)^* \to \mathbb{R}$. The reward is now a real-valued function over finite state-action sequences. Given a (possibly infinite) trace π, the *value* of π is:

$$v(\pi) = \sum_{i=1}^{|\pi|} \gamma^{i-1} R(\langle \pi(1), \pi(2), \ldots, \pi(i)\rangle),$$

where $0 < \gamma \le 1$ is the discount factor and $\pi(i)$ denotes the pair (s_{i-1}, a_i). Since every policy $\rho : S^* \to A$ induces a distribution over the set of possible infinite traces, we can now define the value of a policy ρ given an initial state s_0 to be

$$v^\rho(s) = E_{\pi \sim \mathcal{M},\rho,s_0} v(\pi)$$

That is, $v^\rho(s)$ is the expected value of infinite traces, where traces are distributed according to the distribution over traces defined by the initial state s_0, the transition function Tr, and the policy ρ.

Specifying a non-Markovian reward function explicitly is cumbersome and unintuitive, even if we only want to reward a finite number of traces. But, typically, we want to reward behaviors that correspond to various patterns. LDL$_f$ provides us with an intuitive and convenient language for specifying R implicitly, using a set of pairs $\{(\varphi_i, r_i)_{i=1}^m\}$. Intuitively, if the current trace is $\pi = \langle s_0, a_1, \ldots, s_{n-1}, a_n\rangle$, the agent receives at s_n a reward r_i for every formula φ_i satisfied by π. Formally:

$$R(\pi) = \sum_{1 \le i \le m : \pi \models \varphi_i} r_i$$

From now on, we shall assume that R is thus specified.

Coming back to our running example, we could have the following formulas φ_i, each associated with reward r_i:

$\varphi_1 = [true^*](request_p \to \langle true^*\rangle coffee_p)$

$\varphi_2 = [true^*]([openDoor_d] closeDoor_d)$

$\varphi_3 = \langle((\neg restrd_a)^*; permission_a; (\neg restrd_a)^*; restrd_a)^*; (\neg restrd_a)^*\rangle end$

4 Building an Equivalent Markovian Model

When the rewards are Markovian, one can compute v^ρ (for stationary ρ) and an optimal policy ρ^* using Bellman's dynamic programming equations [Puterman, 2005]. However, this is not the case when the reward is non-Markovian and the policy is non-stationary. The standard solution for this problem is to formulate an extended MDP in which the rewards are Markovian, that is *equivalent* to the original NMRDP [Bacchus *et al.*, 1996; Thiébaux *et al.*, 2006].

Definition 1 ([Bacchus *et al.*, 1996])**.** *An NMRDP $\mathcal{M} = \langle S, A, Tr, R\rangle$ is equivalent to an extended MDP $\mathcal{M}' = \langle S', A, Tr', R'\rangle$ if there exist two functions $\tau : S' \to S$ and $\sigma : S \to S'$ such that*

1. *$\forall s \in S : \tau(\sigma(s)) = s$;*
2. *$\forall s_1, s_2 \in S$ and $s'_1 \in S'$: if $Tr(s_1, a, s_2) > 0$ and $\tau(s'_1) = s_1$, there exists a unique $s'_2 \in S'$ such that $\tau(s'_2) = s_2$ and $Tr(s'_1, a, s'_2) = Tr(s_1, a, s_2)$;*
3. *For any feasible trajectory $\langle s_0, a_1, \ldots, s_{n-1}, a_n\rangle$ of $\mathcal{M}$ and $\langle s'_0, a_1, \ldots, s'_{n-1}, a_n\rangle$ of $\mathcal{M}'$, such that $\tau(s'_i) = s_i$ and $\sigma(s_0) = s'_0$, we have $R(\langle s_0, a_1, \ldots, s_{n-1}, a_n\rangle) = R'(\langle s'_0, a_1, \ldots, s'_{n-1}, a_n\rangle)$.*

As in previous work, we restrict our attention to extended MDPs such that $S' = Q \times S$, for some set Q.

Given an NMRDP $\mathcal{M} = \langle S, A, Tr, R\rangle$, we now show how to construct an equivalent extended MDP. For each reward formula φ_i, we consider the corresponding (minimal) DFA $\mathcal{A}_{\varphi_i} = (2^{\mathcal{P}}, Q_i, q_{i0}, \delta_i, F_i)$, where:

- $2^{\mathcal{P}}$ is the input alphabet of the automaton;
- Q_i is the finite set of states;
- $q_{i0} \in Q_i$ is the initial state;
- $\delta_i : Q \times 2^{\mathcal{P}} \to Q$ is the deterministic transition function (which is total);
- $F_i \subseteq Q_i$ are the accepting states.

We now define the equivalent extended MDP $\mathcal{M}' = \langle S', A', Tr', R'\rangle$ where:

- $S' = Q_1 \times \cdots \times Q_m \times S$ is the set of states;
- $A' = A$;
- $Tr' : S' \times A' \times S' \to [0, 1]$ is defined as follows:
$$Tr'(q_1, \ldots, q_m, s, a, q'_1, \ldots, q'_m, s') = \begin{cases} Tr(s, a, s') & \text{if } \forall i : \delta(q_i, s) = q'_i \\ 0 & \text{otherwise;} \end{cases}$$
- $R' : S' \times A \to \mathbb{R}$ is defined as:
$$R(q_1, \ldots, q_m, s, a) = \sum_{i : \delta(q_i, s) \in F_i} r_i$$

That is, the state space is a product of the states of the original MDP and the various automata. The actions set is the same. Given action a, the s component of the state progresses according to the original MDP dynamics, and the other components progress according to the transition function of the corresponding automata. Finally, in every state, and for every $1 \le i \le m$, the agent receives the reward associated with φ_i if the FSA $\mathcal{A}_{\varphi_i}$ reached a final state.

Theorem 3. *The NMRDP $\mathcal{M} = \langle S, A, Tr, R\rangle$ is equivalent to the extended MDP $\mathcal{M}' = \langle S', A', Tr', R'\rangle$.*

Proof. Recall that every $s' \in S'$ has the form $(q_1, \ldots, q_m, s)$. Define $\tau(q_1, \ldots, q_m, s) = s$. Define $\sigma(s) = (q_{10}, \ldots, q_{m0}, s)$. We have $\tau(\sigma(s)) = s$. Condition 2 is easily verifiable by inspection. For condition 3, consider a possible trace $\pi = \langle s_0, a_1, \ldots, s_{n-1}, a_n\rangle$. We use σ to obtain $s'_0 = \sigma(s_0)$ and given s_i, we define s'_i (for $1 \le i < n$) to be the unique state $(q_{1i}, \ldots, q_{mi}, s_i)$ such that $q_{ji} = \delta_i(q_{ji-1}, a_i)$ for all $1 \le j \le m$. We now have a corresponding possible trace of $\mathcal{M}'$, i.e., $\pi' = \langle s'_0, a_1, s'_1 \ldots, s'_{n-1}, a_n\rangle$. This is the only feasible trajectory of $\mathcal{M}'$ that satisfies Condition 3. The reward at $\pi = \langle s_0, a_1, s_1 \ldots, s_{n-1}, a_n\rangle$ depends only on whether or not each formula φ_i is satisfied by π. However, by construction of the automaton $\mathcal{A}_{\varphi_i}$ and the transition function Tr', $\pi \models \varphi_i$ *iff* $s'_{n-1} = (q_1, \ldots, q_m, s'_n)$ and $q_i \in F_i$. □

Let ρ' be a policy for the Markovian $\mathcal{M}'$. It is easy to define an equivalent policy on $\mathcal{M}$: Let $\pi = \langle s_0, a_1, s_1 \ldots, s_{n-1}, a_n\rangle$ be the current history of the process. Let q_{in} denote the current state of automaton $\mathcal{A}_{\varphi_i}$ given input π. Define $\rho(\pi) := \rho'(q_{1n}, \ldots, q_{mn}, s_n)$.

Theorem 4 ([Bacchus *et al.*, 1996]). *Given an NMRDP $\mathcal{M}$, let ρ' be an optimal policy for an equivalent MDP $\mathcal{M}'$. Then, policy ρ for $\mathcal{M}$ that is equivalent to ρ' is optimal for $\mathcal{M}$.*

5 Minimality and Compositionality

One advantage of our construction is that it benefits from two types of minimality and from compositionality. The Markovian model is obtained by taking the synchronous product of the original MDP and an FSA that is itself the synchronous product of smaller FSAs, one for each formula. We can apply the simple, standard automaton minimization algorithm to obtain a minimal automaton, thus obtaining a minimal MDP. But even better, as we show below, it is enough to ensure that each FSA $\mathcal{A}_{\varphi_i}$ in the above construction is minimal to ensure the overall minimality of the extended MDP.

Theorem 5. *If every automaton $\mathcal{A}_{\varphi_i}$ ($1 \le i \le m$) is minimal then the extended MDP defined above is minimal.*

Proof. Let $\mathcal{A}_s$ be the synchronous product of $\mathcal{A}_{\varphi_i}$ ($1 \le i \le m$). We show that no two distinct states of the synchronous product $\mathcal{A}_s$ are equivalent, and therefore, all of them are needed, hence the thesis.

Suppose that there are two distinct states of the synchronous product $\mathcal{A}_s$ that are equivalent. Then, being $\mathcal{A}_s$ a DFA, such two states are bisimilar. Two states of $\mathcal{A}_s$ are bisimilar (denoted by $\sim$) iff: $(q_1, \ldots, q_n) \sim (t_1, \ldots, t_m)$ implies

- for all i. $q_i \in F_i$ iff $t_i \in F_i$;
- for all a. $\delta_s(q_1, \ldots, q_m) = (q'_1, \ldots, q'_m)$ implies $\delta_s(t_1, \ldots, t_m, a) = (t'_1, \ldots, t'_m)$ and $(q'_1, \ldots, q'_m) \sim (t'_1, \ldots, t'_m)$;
- for all a. $\delta_s(t_1, \ldots, t_m) = (t'_1, \ldots, t'_m)$ implies $\delta_s(q_1, \ldots, q_m, a) = (q'_1, \ldots, q'_m)$ and $(q'_1, \ldots, q'_m) \sim (t'_1, \ldots, t'_m)$.

Now we show that $(q_1, \ldots, q_m) \sim (t_1, \ldots, t_m)$ implies $q_i = t_i$, for all i. To check this we show that the relation "project on i", $\Pi_i((q_1, \ldots, q_m) \sim (t_1, \ldots, t_m))$ extracting the i-th component on the left and on the right of $\sim$ is a bisimulation for states in $\mathcal{A}_i$. Indeed it is immediate to verify that $\Pi_i((q_1, \ldots, q_m) \sim (t_1, \ldots, t_m))$ implies

- $q_i \in F_i$ iff $t_i \in F_i$;
- for all a, $\delta_i(q_i, a) = q'_i$ implies $\delta_i(t_i, a) = t'_i$ and $\Pi_i((q'_1, \ldots, q'_m) \sim (t'_1, \ldots, t'_m))$;
- for all a, $\delta_i(t_i, a) = t'_i$ implies $\delta_i(q_i, a) = q_i$ and $\Pi_i((q'_1, \ldots, q'_m) \sim (t'_1, \ldots, t'_m))$.

Hence if there are two distinct states $(q_1, \ldots, q_m) \sim (t_1, \ldots, t_m)$ then at least for one i it must be the case that q_i and t_i are distinct and bisimilar and hence equivalent. But this is impossible since each DFA $\mathcal{A}_{\varphi_i}$ is minimal. □

In general, if we take the synchronous product of two minimised DFA's we may be able to minimize it further. But since in our case we need to keep the final states of the different DFA's distinct (to assign the proper rewards), as the proof of theorem above shows, no further minimization is possible.

Observe that the above theorem also implies that the construction is compositional, and hence, incremental – if we care for a new formula, we do not need to change the MDP, but simply extend it with one additional component. If the original MDP was minimal and the new component is minimal, then so is the resulting MDP.

[Thiébaux *et al.*, 2006] consider a different minimality criterion, *blind minimality*, which essentially says that, given an initial state s_0 for our NMRDP $\mathcal{M}$, one can construct the set of states reachable from s_0 in $\mathcal{M}'$ without having to generate any unreachable extended state. In particular, this implies that one does not generate the entire automaton for each formula, but construct only its reachable states. Moreover, one can even focus on a subset of reachable states that correspond to trajectories of interest.

We enjoy both notions. We can progress the extended MDP, starting from the initial state, building it on the fly. But we can also start by generating the reachable states of each automaton separately; minimize each automaton, and take their synchronous product. While the automata are theoretically large (as is the reachable state space), in practice, experience shows them to be quite small. Once we have the minimal structure of the automaton, we can progress the extended MDP working with the product of the MDP and automaton states.

6 Getting rewards for complete traces only

We may want to reward an agent for its entire behavior rather than for each prefix of it. This means that the value of a sequence $\pi = \langle s_0, a_1, s_1 \ldots, s_{n-1}, a_n\rangle$ is defined as follows:

$$v(\pi) = \sum_{i:\pi \models \varphi_i} r_i$$

Behaviors optimal w.r.t. this definition will differ from ones that are optimal w.r.t. the original definition in which rewards are collected following each action. The point is that an agent must now attempt to make as many formulas true at once, as it does not get any "credit" for having achieved them in the past.

Given an NMRDP $\mathcal{M}$ with the above reward semantics, we can easily generate an equivalent MDP using the above construction, preceded by the following steps:

1. Add a special action *stop* to A.
2. Add a new proposition *done* to S.
3. No action is applicable in a state in which *done* is *true*.
4. The only effect of the *stop* action is to make *done* be *true*.
5. Convert every reward formula φ_i to $done \wedge \varphi_i$.

Interestingly, when focussing on complete traces, our framework becomes an extension of Goal MDP planning that handles temporally extended goals, see, e.g, Chapter 6 and Chapter 4 of [Geffner and Bonet, 2013].

7 Comparison with previous proposals

Capturing PLTL rewards. The setting proposed can be seen as an extension of [Bacchus *et al.*, 1996]. There, rewards are assigned to partial traces whenever the last state of the trace satisfies a past-LTL (PLTL) formulas. Without introducing explicitly PLTL, but given a partial trace $\pi_0, \ldots, \pi_n$ we reverse it into $\pi_n, \ldots, \pi_0$ and evaluate it over the LTL_f formula φ obtained from the PLTL formula by simply replacing the past operators with the corresponding future operators (e.g., replace *since* with *eventually*). Then, the setting remains analogous to the one shown above.

In particular, we can construct the NFA A_φ associated with φ and, instead of reversing the partial traces, reverse A_φ, thus getting an NFA A_φ^-, by simply reversing the edge directions and switching initial and final states. This can be done in linear time. If we now determinize (and minimize) A_φ^-, getting the (minimal) DFA A_φ^-, we can proceed exactly as above.

Given the above essential equivalence of PLTL and LTL_f, and the fact that LDL_f is strictly more expressive than LTL_f, we conclude that our setting is *strictly more* expressive than the one in [Bacchus *et al.*, 1996]. In principle, we could simply replace PLTL with past-LDL_f. But defining properties in past-LDL_f is likely to be unnatural, since we would have to reverse the regular expressions in the eventualities, and since these have a procedural flavour, it would be somehow like reversing a program. In addition, since the automata construction algorithm is based on progression, unlike [Bacchus *et al.*, 1996], we can use information about the initial state to prevent the generation of unreachable states.

Finally, we note that in [Bacchus *et al.*, 1997], the authors extend their work to handle NMRDPs in factored form and attempt to ensure that the extended state retains this factored form. We note that our construction retains the original form of the MDP, whether factored or not, and generates a natural factored extended state, using one factor per reward formula.

Comparing with $FLTL rewards. In [Thiébaux *et al.*, 2002; Gretton *et al.*, 2003; Thiébaux *et al.*, 2006] a sophisticated temporal logic, called $FLTL is introduced, which is able to specify explicitly when a partial (finite) trace gets a rewards. The exact expressive power of $FLTL has not been assessed yet, and it is open whether it is able to capture PLTL rewards of [Bacchus *et al.*, 1996] and vice-versa. As a result, it remains open to compare our setting, based on LDL_f, with $FLTL. However, as $FLTL is based on LTL, which cannot capture MSO, it would be rather surprising if $FLTL was able to capture the LDL_f rewards proposed here.

We can show, though, how some $FLTL formulas can be expressed in LDL_f[1]. We consider the examples of [Thiébaux *et al.*, 2006]. Some of these show how the PLTL formulas used in [Bacchus *et al.*, 1996] can be expressed in $FLTL. By the relationship between PLTL and LDL_f discussed above, it is immediate that these admit an LDL_f equivalent formula. For instance, $\neg p\mathcal{U}(p \wedge \$)$, which rewards only the first time p is achieved, is equivalent to the PLTL formula $p \wedge \neg\ominus\diamondminus p$, which can be, in turn, rewritten in LDL_f as $\langle \neg p^*; p\rangle end$ or in LTL_f syntax as $\neg p\mathcal{U}(p \wedge last)$. There are also $FLTL formulas for which no equivalent in PLTL is reported. This is the case, e.g., of $\neg q\mathcal{U}((\neg p \wedge \neg q) \vee (q \wedge \$))$, which rewards the holding of p until the occurrence of q, and whose LDL_f translation is $\langle q^*; p\rangle end$ or, in LTL_f, $p\mathcal{U}(q \wedge last)$. Observe how simpler it is to have an intuition of the semantics when using the LDL_f version compared with the $FLTL one. In particular, the latter requires a rigorous application of the semantics even to simply check that the property is as claimed above. In addition, various properties compactly expressible in LDL_f require much more sophisticated encoding when only standard temporal operators are used. Finally, computationally, we can offer the benefits of true minimality and blind minimality (reachability) as well as compositionality.

8 Conclusion

We presented a new language for specifying non-Markovian rewards in MDPs. Our language is more expressive than previous proposals and being based on a standard temporal logics of the future, is likely to be more intuitive to use. We showed how to construct a minimal equivalent MDP, and since we rely on general methods for tracking temporal formulas, the construction is cleaner. Being based on progression, it can use information about the initial state to prune unreachable states.

One problem with non-Markovian rewards is that the reward is only obtained when the entire sequence satisfies the property. This is especially true if we wish to give rewards for complete traces only. In that case, the reward comes as a "surprise" when the last action $stop$ is chosen. We can help solution algorithms if we can start rewarding such behaviors even before the formula is satisfied, helping to guide both search and learning algorithms towards better behaviors. In future work we intend to examine the use of monitoring notions developed for LTL_f and LDL_f [Bauer *et al.*, 2010; De Giacomo *et al.*, 2014; Maggi *et al.*, 2011]. Using such monitors one could extract early rewards that guide the process to get full rewards later.

Another important direction for future work is exploiting non-Markovian rewards in reinforcement learning (RL) to provide better guidance to the learning agent, as well as extending inverse RL methods to learn to assign non-Markovian rewards in a state. We are currently exploring this latter issue when the set of formulas φ_i is given, but the associated reward r_i is unknown.

[1] For further examples see `tinyurl.com/j6l8guj`

References

[Bacchus *et al.*, 1996] Fahiem Bacchus, Craig Boutilier, and Adam J. Grove. Rewarding behaviors. In *Proc. of AAAI*, pages 1160–1167, 1996.

[Bacchus *et al.*, 1997] Fahiem Bacchus, Craig Boutilier, and Adam J. Grove. Structured solution methods for non-markovian decision processes. In *Proc. of AAAI*, pages 112–117, 1997.

[Bauer *et al.*, 2010] Andreas Bauer, Martin Leucker, and Christian Schallhart. Comparing LTL semantics for runtime verification. *Logic and Computation*, 2010.

[Bonet and Geffner, 2003] B. Bonet and H. Geffner. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *ICAPS'03*, 2003.

[Camacho *et al.*, 2017] Alberto Camacho, Eleni Triantafillou, Christian Muise, Jorge A. Baier, and Sheila McIlraith. Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces. In *Proc. of AAAI*, 2017.

[De Giacomo and Vardi, 2013] Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proc. of IJCAI*, 2013.

[De Giacomo and Vardi, 2015] Giuseppe De Giacomo and Moshe Y. Vardi. Synthesis for LTL and LDL on finite traces. In *Proc. of IJCAI*, 2015.

[De Giacomo and Vardi, 2016] Giuseppe De Giacomo and Moshe Y. Vardi. Ltl_f and ldl_f synthesis under partial observability. In *Proc. of IJCAI*, pages 1044–1050, 2016.

[De Giacomo *et al.*, 2014] Giuseppe De Giacomo, Riccardo De Masellis, Marco Grasso, Fabrizio Maria Maggi, and Marco Montali. Monitoring business metaconstraints based on LTL and LDL for finite traces. In *Proc. of BPM*, 2014.

[Fischer and Ladner, 1979] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18, 1979.

[Geffner and Bonet, 2013] Hector Geffner and Blai Bonet. *A Coincise Introduction to Models and Methods for Automated Planning*. Morgan&Claypool, 2013.

[Gretton *et al.*, 2003] Charles Gretton, David Price, and Sylvie Thiébaux. Implementation and comparison of solution methods for decision processes with non-markovian rewards. In *Proc. of UAI*, pages 289–296, 2003.

[Hansen and Zilberstein, 2001] E. A. Hansen and S. Zilberstein. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129(1-2):35–62, 2001.

[Harel *et al.*, 2000] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. MIT Press, 2000.

[Maggi *et al.*, 2011] Fabrizio Maria Maggi, Marco Montali, Michael Westergaard, and Wil M. P. van der Aalst. Monitoring business constraints with linear temporal logic: An approach based on colored automata. In *Proc. of BPM*, 2011.

[Pnueli, 1977] Amir Pnueli. The temporal logic of programs. In *Proc. of FOCS*, pages 46–57. IEEE Computer Society, 1977.

[Puterman, 2005] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 2005.

[Rabin and Scott, 1959] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. Res. Dev.*, 3:114–125, April 1959.

[Tabakov and Vardi, 2005] Deian Tabakov and Moshe Y. Vardi. Experimental evaluation of classical automata constructions. In *Proc. of LPAR*, 2005.

[Thiébaux *et al.*, 2002] Sylvie Thiébaux, Froduald Kabanza, and John K. Slaney. Anytime state-based solution methods for decision processes with non-markovian rewards. In *Proc. of UAI*, pages 501–510, 2002.

[Thiébaux *et al.*, 2006] Sylvie Thiébaux, Charles Gretton, John K. Slaney, David Price, and Froduald Kabanza. Decision-theoretic planning with non-markovian rewards. *J. Artif. Intell. Res. (JAIR)*, 25:17–74, 2006.

[Torres and Baier, 2015] Jorge Torres and Jorge A. Baier. Polynomial-time reformulations of LTL temporally extended goals into final-state goals. In *Proc. of IJCAI*, pages 1696–1703, 2015.

[Vardi, 2011] Moshe Y. Vardi. The rise and fall of linear time logic. In *Proc. of GandALF*, 2011.