LTL$_f$/LDL$_f$ Non-Markovian Rewards

Ronen I. Brafman
Ben-Gurion University, Beer-Sheva, Israel
brafman@cs.bgu.ac.il

Giuseppe De Giacomo & Fabio Patrizi
Sapienza Università di Roma, Italy
{degiacomo,patrizi}@dis.uniroma1.it

Abstract

In Markov Decision Processes (MDPs), the reward obtained in a state is Markovian, i.e., depends on the last state and action. This 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 non-Markovian reward functions was the subject of two previous lines of work. Both use LTL variants to specify the reward function and then compile the new model back into a Markovian model. Building on recent progress in 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.

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 it was executed in, and that a reward given at a state depends only on the previous action and state. It has long been observed (Bacchus, Boutilier, and Grove 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 ensure it will *access restricted areas only after having acquired the right permission*. Such rewards are non-Markovian. Interestingly, Littman has advocated at IJCAI 2015 that it may actually be more convenient, from a design perspective, to assign rewards to the satisfaction of (non-Markovian) declarative temporal properties, rather than to states (Littman et al. 2017).

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, Boutilier, and Grove 1996) suggest using a temporal logic of the past. Whether a state satisfies such a past temporal formula depends on the entire sequence of states leading to the state. 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 not often used in other areas, and its semantics is more involved.

Existing MDP solution methods, possibly with the exception of Monte-Carlo tree search algorithms (Kocsis and Szepesvári 2006), 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, if we extend our state to record whether a "bring-coffee" command was issued earlier, a reward for bringing coffee in states indicating that "bring-coffee" was issued in the past, is now Markovian. It rewards the same behaviors as the 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 extended MDPs 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, Boutilier, and Grove 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 a 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.

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 MDPs with non-Markovian rewards. With these tools, which were unavailable to earlier work, we are able to provide a cleaner, more elegant approach that builds on well understood semantics, much more expressive languages, and enjoys good algorithmic properties. 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 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; 2015; De Giacomo and Vardi 2016; Torres and Baier 2015; Camacho et al. 2017c). We exploit such techniques to generate an extended MDP with good properties.[1]

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, Boutilier, and Grove 1996), and all examples of (Thiébaux et al. 2006). Moreover, LDL_f can naturally represent "procedural constraints" (Baier et al. 2008), i.e., sequencing constraints expressed as programs, using "if" and "while". In fact, 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 added, we only need to optimize *its* automaton and add it to the current (extended) MDP. If the current MDP was minimal, the resulting (extended) MDP is minimal too. 3. **Forward Construction via Progression.** The automaton used to identify when a reward should be given can be constructed in a forward manner using progression. This ensures the generation of reachable states only, as in (Thiébaux et al. 2006). Moreover, we can combine progression in the space of MDPs with precomputed automata minimization to obtain the best of both worlds.

[1] We share this objective with work independently carried out in (Camacho et al. 2017b; 2017a).

Background

MDPs. A Markov Decision Process (MDP) $\mathcal{M} = \langle S, A, Tr, R\rangle$ contains a set S of states, a set A of actions, and 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 $s \vDash \phi$.

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 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 $s \in S$. When the horizon is infinite, there exists an optimal policy $\rho^* : S \to A$ that is stationary and deterministic (i.e., ρ^* depends only on the current state) (Puterman 2005). There are diverse methods for computing an optimal policy. With the exception of online simulation-based methods, they rely on the Markov assumption, and their theoretical and practical complexity is strongly impacted by $|S|$.

LTL_f and LDL_f. LTL_f is essentially the standard Linear-time Temporal Logic LTL (Pnueli 1977) interpreted over finite, instead of infinite, traces (De Giacomo and Vardi 2013). LTL_f is as expressive as FO over finite traces and star-free regular expressions (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, Kozen, and Tiuryn 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:

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

where tt stands for logical true; ϕ is a propositional formula over $\mathcal{P}$ (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 $[\varrho]\varphi \doteq \neg\langle\varrho\rangle\neg\varphi$ as in PDL, $ff \doteq \neg tt$ for false, and $\phi \doteq \langle\phi\rangle tt$ to denote occurence of propositional formula ϕ.

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.

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 $\pi(i,j) \doteq \pi_i, \pi_{i+1}, \ldots, \pi_{j-1}$. When $j > length(\pi)$, $\pi(i,j) \doteq \pi(i, length(\pi))$.

In decision processes, traces are usually sequences of states *and* actions, i.e., they have the form: $\langle s_0, a_1, s_1, \ldots, s_{n-1}, a_n \rangle$. These can still be represented as traces of the form $\pi = \pi_0, \ldots, \pi_n$, 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 $\pi, i \models \varphi_2$;
- $\pi, i \models \langle\varrho\rangle\varphi$ iff there exists $i \leq j$ such that $\pi(i,j) \in \mathcal{L}(\varrho)$ and $\pi, j \models \varphi$, where the relation $\pi(i,j) \in \mathcal{L}(\varrho)$ is as follows:
  - $\pi(i,j) \in \mathcal{L}(\phi)$ if j=i+1, $i < length(\pi)$, and $\pi(i) \models \phi$ (ϕ propositional);
  - $\pi(i,j) \in \mathcal{L}(\varphi?)$ if $j = i$ and $\pi, i \models \varphi$;
  - $\pi(i,j) \in \mathcal{L}(\varrho_1+\varrho_2)$ if $\pi(i,j) \in \mathcal{L}(\varrho_1)$ or $\pi(i,j) \in \mathcal{L}(\varrho_2)$;
  - $\pi(i,j) \in \mathcal{L}(\varrho_1;\varrho_2)$ if there exists $k \in [i,j]$ such that $\pi(i,k) \in \mathcal{L}(\varrho_1)$ and $\pi(k,j) \in \mathcal{L}(\varrho_2)$;
  - $\pi(i,j) \in \mathcal{L}(\varrho^*)$ if $j = i$ or there exists k such that $\pi(i,k) \in \mathcal{L}(\varrho)$ and $\pi(k,j) \in \mathcal{L}(\varrho^*)$.

Note that if $i \geq length(\pi)$, the above definitions still apply; though, $\langle\phi\rangle\varphi$ (ϕ prop.) and $\langle\varrho\rangle\varphi$ become 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 $\bigcirc\varphi \doteq \langle true\rangle(\varphi \wedge \neg end)$ and $\varphi_1 \,\mathcal{U}\, \varphi_2 \doteq \langle(\varphi_1?; true)^*\rangle(\varphi_2 \wedge \neg end)$, and any RE r, with the formula $\langle r\rangle end$, where $end \doteq [true]ff$ expresses that the trace has ended. Note that in addition to end we can also denote the last element of the trace as $last \doteq \langle true\rangle end$. Section Non-Markovian Rewards illustrates several examples of LDL_f formulas in our context.

Computing DFA for LDL_f formulas

As standard, an NFA is a tuple $\mathcal{A} = \langle \Sigma, Q, q_0, \delta, F\rangle$, where: (i) Σ is the input alphabet; (ii) Q is the finite set of states; (iii) $q_0 \in Q$ is the initial state; (iv) $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation; (v) $F \subseteq Q$ is the set of final states. A DFA is an NFA where δ is a function $\delta : Q \times \Sigma \to Q$. By $\mathcal{L}(\mathcal{A})$ we mean the set of all traces over Σ accepted by $\mathcal{A}$.

We can associate each LDL_f formula φ with an (exponentially large) NFA $\mathcal{A}_\varphi = \langle 2^{\mathcal{P}}, Q, q_0, \delta, F\rangle$ that accepts exactly the traces satisfying φ. A direct algorithm (LDL_f2NFA) for computing the NFA given the LDL_f formula, which is a variant of that in (De Giacomo and Vardi 2015), 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) 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 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). Let ∂ be the following auxiliary function, which takes as input an (implicitly *quoted*) LDL_f formula φ (in NNF), extended with auxiliary constructs $\boldsymbol{F}_\psi$ and $\boldsymbol{T}_\psi$, and a propositional interpretation Θ for $\mathcal{P}$, and returns a positive boolean formula whose atoms are (implicitly quoted) φ subformulas (not including $\boldsymbol{F}_\psi$ or $\boldsymbol{T}_\psi$):

$$
\begin{array}{rcl}
\partial(tt,\Theta) & = & true \\
\partial(ff,\Theta) & = & false \\
\partial(\phi,\Theta) & = & \partial(\langle\phi\rangle tt,\Theta) \quad (\phi \text{ prop.}) \\
\partial(\varphi_1 \wedge \varphi_2,\Theta) & = & \partial(\varphi_1,\Theta) \wedge \partial(\varphi_2,\Theta) \\
\partial(\varphi_1 \vee \varphi_2,\Theta) & = & \partial(\varphi_1,\Theta) \vee \partial(\varphi_2,\Theta) \\
\partial(\langle\phi\rangle\varphi,\Theta) & = & \begin{cases} \boldsymbol{E}(\varphi) \text{ if } \Theta \models \phi \quad (\phi \text{ prop.}) \\ false \text{ if } \Theta \not\models \phi \end{cases} \\
\partial(\langle\varrho?\rangle\varphi,\Theta) & = & \partial(\varrho,\Theta) \wedge \partial(\varphi,\Theta) \\
\partial(\langle\varrho_1+\varrho_2\rangle\varphi,\Theta) & = & \partial(\langle\varrho_1\rangle\varphi,\Theta) \vee \partial(\langle\varrho_2\rangle\varphi,\Theta) \\
\partial(\langle\varrho_1;\varrho_2\rangle\varphi,\Theta) & = & \partial(\langle\varrho_1\rangle\langle\varrho_2\rangle\varphi,\Theta) \\
\partial(\langle\varrho^*\rangle\varphi,\Theta) & = & \partial(\varphi,\Theta) \vee \partial(\langle\varrho\rangle\boldsymbol{F}_{\langle\varrho^*\rangle\varphi},\Theta) \\
\partial([\phi]\varphi,\Theta) & = & \begin{cases} \boldsymbol{E}(\varphi) \text{ if } \Theta \models \phi \quad (\phi \text{ prop.}) \\ true \text{ if } \Theta \not\models \phi \end{cases} \\
\partial([\varrho?]\varphi,\Theta) & = & \partial(nnf(\neg\varrho),\Theta) \vee \partial(\varphi,\Theta) \\
\partial([\varrho_1+\varrho_2]\varphi,\Theta) & = & \partial([\varrho_1]\varphi,\Theta) \wedge \partial([\varrho_2]\varphi,\Theta) \\
\partial([\varrho_1;\varrho_2]\varphi,\Theta) & = & \partial([\varrho_1][\varrho_2]\varphi,\Theta) \\
\partial([\varrho^*]\varphi,\Theta) & = & \partial(\varphi,\Theta) \wedge \partial([\varrho]\boldsymbol{T}_{[\varrho^*]\varphi},\Theta) \\
\partial(\boldsymbol{F}_\psi,\Theta) & = & false \\
\partial(\boldsymbol{T}_\psi,\Theta) & = & true
\end{array}
$$

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 $\partial(\varphi,\epsilon)$ is defined inductively exactly as above, except for the following base cases:

$$\partial(\langle\phi\rangle\varphi,\epsilon) = false \quad \partial([\phi]\varphi,\epsilon) = true \quad (\phi \text{ prop.})$$

Note that $\partial(\varphi,\epsilon)$ is always either $true$ or $false$.

The NFA $\mathcal{A}_\varphi$ for an LDL_f formula φ is then built in a forward fashion as shown in Figure 1, where: states of $\mathcal{A}_\varphi$ are sets of atoms (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 $q' \models \bigwedge_{(\psi\in q)} \partial(\psi,\Theta)$ (notice that we trivially have $(\varnothing, a, \varnothing) \in \delta$ for every $a \in 2^{\mathcal{P}}$).

Theorem 1. *(De Giacomo and Vardi 2015) Algorithm* LDL_f2NFA *is correct, i.e., for every finite trace* π*:* $\pi \models \varphi$ *iff* $\pi \in \mathcal{L}(\mathcal{A}_\varphi)$*. Moreover, it 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*.*

The NFA $\mathcal{A}_\varphi$ can be transformed into a DFA, in exponential time, following the standard procedure, and then possibly put in (the unique) minimal form, in polynomial time (Rabin

algorithm LDL$_f$2NFA
input LDL$_f$ formula φ
output NFA $\mathcal{A}_\varphi = (2^{\mathcal{P}}, Q, q_0, \delta, F)$
$q_0 \leftarrow \{\varphi\}$
$F \leftarrow \{\varnothing\}$
if $(\partial(\varphi, \epsilon) = true)$ **then**
  $F \leftarrow F \cup \{q_0\}$
$Q \leftarrow \{q_0, \varnothing\}$, $\delta \leftarrow \varnothing$
while (Q or δ change) **do**
  for $(q \in Q)$ **do**
    if $(q' \vDash \bigwedge_{(\psi \in q)} \partial(\psi, \Theta)$ **then**
      $Q \leftarrow Q \cup \{q'\}$
      $\delta \leftarrow \delta \cup \{(q, \Theta, q')\}$
      if $(\bigwedge_{(\psi \in q')} \partial(\psi, \epsilon) = true)$ **then**
        $F \leftarrow F \cup \{q'\}$

Figure 1: LDL$_f$2NFA algorithm

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).

Computing the DFA on-the-fly. All operations above can be performed *on-the-fly*, without the need for constructing $\mathcal{A}_\varphi$. To do so, we progress all possible states that the NFA can be in, after consuming the next trace symbol, and accept the trace iff, once it has been completely read, the set of possible states contains a final state. More formally, call a set of possible states for the NFA a *macro* state, let $\mathcal{Q} = \{q_1, \ldots, q_n\}$ be the current macro state (initially $\mathcal{Q} = \mathcal{Q}_0 = \{q_0\} = \{\{\varphi\}\}$), and let Θ be the next trace symbol. Then, the successor macro state is the set $\mathcal{Q}' = \{q' \mid \exists q \in \mathcal{Q} \text{ s.t. } q' \vDash \bigwedge_{(\psi \in q)} \partial(\psi, \Theta)\}$. Given an input trace π, we decide whether $\pi \vDash \varphi$ by iterating the above procedure, starting from the initial state $\mathcal{Q} = \mathcal{Q}_0$, and accepting π iff the last state obtained includes $\{true\}$.

The following observations are in order. Firstly, to compute $\mathcal{Q}'$, only function ∂ is needed. Neither the set of states Q nor the transition relation δ of the NFA are required. In other words, the on-the-fly progression does not require the construction of $\mathcal{A}_\varphi$. Secondly, the progression step produces only one successor macro state $\mathcal{Q}'$, thus transitions are deterministic. Indeed, it is immediate to see that the on-the-fly progression includes the determinization of $\mathcal{A}_\varphi$. As it turns out, the on-the-fly approach performs, in fact, the execution and the determinization at once and in a lazy way, i.e., *avoiding* the construction of the entire resulting DFA.

Non-Markovian Rewards

In this section we extend MDPs with LDL$_f$-based reward functions resulting in a non-Markovian-reward decision process (NMRDP). Specifically, a 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 $0 < \gamma \leq 1$ 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 as:

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

That is, $v^\rho(s)$ is the expected value of infinite traces, where the distribution over traces is 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 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 (partial) 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:

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

From now on, we assume R is thus specified. Note that we use LDL$_f$ to reward partial traces. Sometimes, we may want to reward complete traces only, this is studied later.

Examples. To illustrate the power of LDL$_f$ as a mechanism to specify non-Markovian rewards, we show some examples.

The properties mentioned in the introduction are LDL$_f$-expressible: $[true^*](request_p \to \langle true^*\rangle coffee_p)$ (all coffee requests from person p will eventually be served); $\langle((\neg restr_a)^*; perm_a; (\neg restr_a)^*; restr_a)^*; (\neg restr_a)^*\rangle end$ (before entering restricted area a the robot must have permission for a). Note that the first formula can be easily rewritten in LTL$_f$, as $\Box(request_p \to \Diamond coffee_p)$, but not the second one.

Also the formulas from (Bacchus, Boutilier, and Grove 1996) and (Thiébaux et al. 2006) are LDL$_f$-expressible: 1. $\langle \neg g^*; g\rangle end$, reward offered only at the first state where g holds; 2. $\langle true^*; g; true^*\rangle end$, reward offered at every state that follows g (included); 3. $\langle \neg g^*; g; (\neg g^k; \neg g^*; g)^*)\rangle end$, achievement of g is rewarded periodically, at most once every k steps; 4. $\langle true^*; \neg g; g + ((\neg g^1 + \ldots + \neg g^k); g)\rangle end$, achievement of g is rewarded whenever it occurs within k steps ($k \geq 1$) of a state where $\neg g$ holds; 5. $\langle true^*; g_1; g_2; g_3\rangle end$, reward issued whenever g_1 is achieved and followed immediately by g_2 and then by g_3; 6. $\langle true^*; c; true^*; g\rangle end$ achievement of g is rewarded whenever it follows c; 7. $\langle true^*; c; \neg g; \neg g^*; g\rangle end$, only the first achievement of g that follows c is rewarded; 8. $\langle true^*; c; g\rangle end$, g is rewarded whenever it follows c immediately; 9. $\langle true^*; c; g + ((true^1 + \ldots + true^k); g)\rangle end$, achievement of g is rewarded whenever occurring within k steps ($k \geq 1$) of c; 10. $\langle true^*; c; g + ((\neg g^1 + \ldots + \neg g^k); g)\rangle end$, only the first achievement of g occurring within k steps ($k \geq 1$) of c is rewarded; 11. $\langle g^*\rangle end$, reward issued if g has always been true; 12. $\langle c^*; g\rangle end$, the holding of c until g is rewarded.

To appreciate the power of LDL$_f$, consider the following example. Suppose one prefers policies for a physician with the following structure (on top of whatever other requirements/rewards exist): the physician should work in clear phases: check patient (a) then update patient record (b) repeatedly. Occasionally, and always in the end, after treating

a patient and updating her record, upload updated records to server (c). This can be concisely captured in LDL$_f$ by $\langle((a;b)^*;c)^*\rangle end$. The equivalent LTL$_f$ formula would be:

$$\begin{array}{l}(\neg b\,\mathcal{W}\,a)\wedge\\ \Box(b\rightarrow\circ(\neg b\,\mathcal{W}\,a))\wedge\\ \Box((c\wedge\circ true)\rightarrow\Diamond c)\wedge\\ \Box(a\rightarrow\circ b)\wedge\\ \Box(b\rightarrow\circ(c\vee a))\end{array}$$

If a,b and c are non exclusive formulas (they can be true simultaneously), the equivalent LTL$_f$ formula can be much more complex (depending on the actual formulas a, b, c), and in some cases it may not exist at all (as in the limit case where $a = b = c = true$).

LDL$_f$, differently from LTL$_f$, can also easily express *procedural constraints* (De Giacomo and Vardi 2015; Fritz and McIlraith 2007; Baier et al. 2008), so one can reward the traces satisfying such constraints. To see this, consider a sort of propositional variant of GOLOG (Levesque et al. 1997):

$$\begin{array}{rcl}\varrho & ::= & A \mid \phi? \mid \varrho_1 + \varrho_2 \mid \varrho_1;\varrho_2 \mid \varrho^* \mid \quad (A \text{ prop.})\\ & & \textbf{if } \phi \textbf{ then } \varrho_1 \textbf{ else } \varrho_2 \mid \textbf{while } \phi \textbf{ do } \varrho,\end{array}$$

These programs correspond to LDL$_f$ path expressions, with **if** and **while** abbreviations (Fischer and Ladner 1979):

$$\begin{array}{rcl}\textbf{if } \phi \textbf{ then } \varrho_1 \textbf{else } \varrho_2 & \doteq & (\phi?;\varrho_1) + (\neg\phi?;\varrho_2)\\ \textbf{while } \phi \textbf{ do } \varrho & \doteq & (\phi?;\varrho)^*;\neg\phi?\end{array}$$

We can reward the traces that follow such (nondeterministic) programs. E.g.: *At every point, if it is hot then, if the air-conditioning is off, turn it on, else do not turn it off*:

$$\begin{array}{l}[true^*]\langle\textbf{if } (hot) \textbf{ then}\\ \qquad\qquad \textbf{if } (\neg airOn) \textbf{ then } turnOnAir\\ \qquad\qquad \textbf{else } \neg turnOffAir\rangle true\end{array}$$

As another example: *Alternate the following two instructions: while it is hot, if the air-conditioning is off then turn it on, else do not turn it off; do something for one step*:

$$\begin{array}{l}\langle(\textbf{while } (hot) \textbf{ do}\\ \quad \textbf{if } (\neg airOn) \textbf{ then } turnOnAir\\ \quad \textbf{else } \neg turnOffAir; true)^*\rangle end\end{array}$$

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 thus the optimal policy may not be stationary. The standard solution is to formulate an extended MDP (with Markvian rewards) that is *equivalent* to the original NMRDP (Bacchus, Boutilier, and Grove 1996; Thiébaux et al. 2006).

Definition 1 ((Bacchus, Boutilier, and Grove 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. First, using the methods described earlier, construct for each reward formula φ_i its corresponding (minimal) DFA, $\mathcal{A}_{\varphi_i} = \langle 2^{\mathcal{P}}, Q_i, q_{i0}, \delta_i, F_i\rangle$ (notice that $S \subseteq 2^{\mathcal{P}}$ and δ_i is total).

Then, 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:

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

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

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

That is, the state space is a product of the states of the original MDP and the various automata. The action 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 automaton. Finally, in every state, and for every $1 \le i \le m$, the agent receives the reward associated with φ_i if the DFA $\mathcal{A}_{\varphi_i}$ reached a final state.

Theorem 2. *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 $\delta(q_1, \ldots, q_m, s) = s$. Define $\sigma(s) = (q_{10}, \ldots, q_{m0}, s)$. We have $\delta(\sigma(s)) = s$. Condition 2 of Def. 1 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 $q_{ji} = \delta(q_{ji-1}, a_i)$ 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', $\pi \vDash \varphi_i$ *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 leading to state s_n. 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 3 ((Bacchus, Boutilier, and Grove 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 $\mathcal{M}$.*

Minimality and Compositionality

One of the main aims of previous work on NMRDP specification methods was to help minimize the size of the resulting MDP. We now explain the attractive minimality properties supported by our construction. Note that the minimization discussed below is w.r.t. the extended MDP. It is quite possible that the original NMRDP can be minimized by removing duplicate states. Its minimization is orthogonal to the issue of minimizing the extended MDP, and hence, we assume that the NMRDP itself is already minimal.

Constructing a Minimal MDP. The Markovian model is obtained by taking the synchronous product of the original MDP and a DFA that is itself the synchronous product of smaller DFA's, 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 DFA $\mathcal{A}_{\varphi_i}$ in the above construction is minimal to ensure the overall minimality of the extended MDP.

Theorem 4. *If every automaton $\mathcal{A}_{\varphi_i}$ $(1 \leq i \leq m)$ is minimal then the extended MDP defined above is minimal.*

Proof. Let $\mathcal{A}_s$ be the synchronous product of $\mathcal{A}_{\varphi_i}$ $(1 \leq i \leq m)$. 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 two distinct states of the synchronous product $\mathcal{A}_s$ 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, the synchronous product of minimal DFA's can be minimized further. The theorem shows that no further minimization is possible if the final states of the automata are kept distinct, to assign proper rewards. This is not required if one does not need to distinguish between multiple reward formulas (in which case, final states can be conjoined).

This theorem also implies that the construction is *compositional* and, hence, incremental: If a new formula is added, we need not change the MDP, but simply extend it with one additional component. If the original MDP was minimal and the new component is minimal, so is the resulting MDP.

Generating Reachable States Only. Minimizing the state space of the extended MDP is important if one wants to apply classical dynamic programming algorithms such as value iteration (Bellman 1957) and policy iteration (Howard 1960). However, these methods typically do not scale up as the size of S increases. Instead, search-based algorithms, such as LAO* (Hansen and Zilberstein 2001), LRTDP (Bonet and Geffner 2003), or MCTS (Kocsis and Szepesvári 2006) are preferred. Their main advantage is that they explore only a subset of the reachable states. These states are generated by progressing the initial state with a sequence of actions. For this reason, (Thiébaux et al. 2006) developed an approach in which progression can be applied to generate the states of the extended MDP, as well. That is, one does not need to enumerate the entire set S of the original NMRDP and the entire set S' of the extended MDP a priori.

We, too, support progression, and in a manner that is simpler to understand and analyze. Recall that the automaton that tracks the satisfaction of a reward formula can be constructed by progression, and that the states of the extended MDP are simply vectors that represent the state of the NMRDP and the state of the automaton for each reward formula. Hence, we can trivially support progression by simply progressing each component of this vector in the standard manner.

As observed by (Thiébaux et al. 2006), when progression is used, the constructed states may not be minimal. That is, one may generate two extended states that have an identical underlying MDP state but a different extended part, yet both states are equivalent in the sense that we get the same reward behavior from both. This issue is easy to understand with our automata-based construction: It is simply a result of the fact that the automaton constructed by progression is not minimal. Our construction provides the user with a clear set of options:

1. Apply simple progression, possibly generating a non-minimal automaton for some of the formulas. This is essentially the approach of (Thiébaux et al. 2006). (But see the discussion in Section .)
2. Build the automata for the reward formulas and minimize them off-line before starting search. Then, apply progression using the minimal automata. This approach is attractive because the reward formulas are typically much smaller than the MDP and so is the size of their automata.

Hence the effort of constructing and minimizing them will not be significant. Yet, by minimizing them a-priori, we can reduce the size of the combined state space (which is a product of the two) significantly.

3. Apply a more complex progression algorithm that generates a minimal automaton on-the-fly in time quadratic in the size of the minimal automata for the reward formulas (Lee and Yannakakis 1992).

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 may differ from those optimal w.r.t. the original definition in which rewards are collected following each action. Now, an agent must 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).

Comparison with previous proposals

Capturing PLTL rewards. Our proposal can be seen as extending (Bacchus, Boutilier, and Grove 1996). There, rewards are assigned to partial traces whenever the last state of the trace satisfies 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 this 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, Boutilier, and Grove 1996). In addition, unlike (Bacchus, Boutilier, and Grove 1996), our automata construction algorithm is based on progression, allowing us to use information about the initial state to prevent the generation of unreachable states.

Comparing with $FLTL rewards. In (Thiébaux, Kabanza, and Slaney 2002; Gretton, Price, and Thiébaux 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 reward. In fact, many formulas in this logic cannot naturally be interpreted as specifying reasonable rewards. Thus, the results of (Thiébaux et al. 2006) focus on a class of formulas called *reward-normal*. This class has been further studied in (Slaney 2005) and in (Gretton 2014). In this work it is first shown that a notable fragment of *reward-perfect* formulas is equivalent in expressive power to star-free RE, and hence equivalent to LTL_f. Then, *reward-normal* formulas are shown to be expressible as RE, and hence expressible in LDL_f. Moreover, in (Gretton 2014), a variant of *reward-normal* formulas is introduced with exactly the same expressive power as RE_f, and hence equivalent to LDL_f. Interestingly, the reduction to RE is based on an actual translation into finite state automata.

Besides the useful ability to build the extended MDP by progression, (Thiébaux et al. 2006) discuss a minimality notion called *blind minimality* where no two states in the extended MDP lead to the same reward behavior under all conceivable futures (where a "conceivable future" is any future sequence of states and actions, including ones that are not reachable). Observe that if one considers an NMRDP with a single state, this amounts to being able to generate the *minimal* automata for the reward formulas by progression. We are unaware of any general method for providing such a construction in linear time, which further hints that the set of formulas for which this result holds is rather limited. Given the above, we cannot claim to provide blind minimality. We can only speculate that given the similarity between their progression algorithm and the standard progression algorithm in cases where they are both defined, we should be able to offer similar guarantees. Thus, the formalism developed here is simpler syntactically and semantically, most likely more expressive, can support minimization without requiring the construction of the NMRDP, but supports progression as well, and is compositional.

Conclusion

We presented a new language for specifying non-Markovian rewards in MDPs. Our language is based on LTL_f/LDL_f and is more expressive than previous proposals and being based on a standard temporal logic 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.

In future work we intend to examine the use of monitoring notions developed for LTL_f and LDL_f (Bauer, Leucker, and Schallhart 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.

Finally, we observe that LDL_f can capture all co-safe LTL formulas, i.e., LTL formulas whose automaton is a DFA (instead of a Büchi automaton). Co-safe LTL formulas were used in Robotics to select traces of an MDP that represent behaviors of interest (Lacerda, Parker, and Hawes 2014; 2015). Thus, instead of co-safe LTL, we can use LDL_f to gain expressivity in trace specification at essentially no cost.

Acknowledgements: The first author is supported in part by ISF grant 933/13 and the Lynn and William Frankel Center for Computer Science. The work was partially supported by the Sapienza project "Immersive Cognitive Environments".

References

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

Baier, J. A.; Fritz, C.; Bienvenu, M.; and McIlraith, S. A. 2008. Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners. In *AAAI*.

Bauer, A.; Leucker, M.; and Schallhart, C. 2010. Comparing LTL semantics for runtime verification. *Logic and Computation*.

Bellman, R. 1957. *Dynamic Programming*. Princeton University Press.

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

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

Camacho, A.; Chen, O.; Sanner, S.; and McIlraith, S. A. 2017b. Non-markovian rewards expressed in LTL: guiding search via reward shaping. In *SOC*, 159–160.

Camacho, A.; Triantafillou, E.; Muise, C.; Baier, J. A.; and McIlraith, S. 2017c. Non-deterministic planning with temporally extended goals: LTL over finite and infinite traces. In *AAAI*.

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

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

De Giacomo, G., and Vardi, M. Y. 2016. LTL_f and LDL_f synthesis under partial observability. In *IJCAI*.

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*.

Fischer, M. J., and Ladner, R. E. 1979. Propositional dynamic logic of regular programs. *J. Com. Systems and Science* 18.

Fritz, C., and McIlraith, S. A. 2007. Monitoring plan optimality during execution. In *ICAPS*.

Geffner, H., and Bonet, B. 2013. *A Coincise Introduction to Models and Methods for Automated Planning*. Morgan&Claypool.

Gretton, C.; Price, D.; and Thiébaux, S. 2003. Implementation and comparison of solution methods for decision processes with non-markovian rewards. In *UAI*.

Gretton, C. 2014. A more expressive behavioral logic for decision-theoretic planning. In *PRICAI*, 13–25.

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

Harel, D.; Kozen, D.; and Tiuryn, J. 2000. *Dynamic Logic*. MIT Press.

Howard, R. A. 1960. *Dynamic Programming and Markov Processes*. MIT Press.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *ECML*.

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

Lacerda, B.; Parker, D.; and Hawes, N. 2015. Optimal Policy Generation for Partially Satisfiable Co-Safe LTL Specifications. In *IJCAI*.

Lee, D., and Yannakakis, M. 1992. Online minimization of transition systems (extended abstract). In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*.

Levesque, H. J.; Reiter, R.; Lesperance, Y.; Lin, F.; and Scherl, R. 1997. GOLOG: A logic programming language for dynamic domains. *J. of Logic Programming* 31.

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.

Maggi, F. M.; Montali, M.; Westergaard, M.; and van der Aalst, W. M. P. 2011. Monitoring business constraints with linear temporal logic: An approach based on colored automata. In *Proc. of BPM*.

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

Puterman, M. L. 2005. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley.

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

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

Tabakov, D., and Vardi, M. Y. 2005. Experimental evaluation of classical automata constructions. In *LPAR*.

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

Thiébaux, S.; Kabanza, F.; and Slaney, J. K. 2002. Anytime state-based solution methods for decision processes with non-markovian rewards. In *UAI*.

Torres, J., and Baier, J. A. 2015. Polynomial-time reformulations of LTL temporally extended goals into final-state goals. In *IJCAI*.

Vardi, M. Y. 2011. The rise and fall of linear time logic. In *GandALF*.