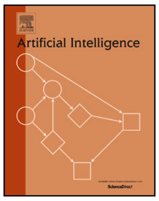
Artificial Intelligence 331 (2024) 104113

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Regular decision processes

Ronen I. Brafman [a,*], Giuseppe De Giacomo [b]

[a] *Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, 84105, Israel*
[b] *Department of Computer Science, University of Oxford, Oxford, OX1 3QD, UK*

ARTICLE INFO

Keywords:
Markov-decision processes
Non-Markovian decision processes
POMDPs
Regular languages

ABSTRACT

We introduce and study Regular Decision Processes (RDPs), a new, compact model for domains with non-Markovian dynamics and rewards, in which the dependence on the past is regular, in the language theoretic sense. RDPs are an intermediate model between MDPs and POMDPs. They generalize k-order MDPs and can be viewed as a POMDP in which the hidden state is a regular function of the entire history. In factored RDPs, transition and reward functions are specified using formulas in linear temporal logics over finite traces, or using regular expressions. This allows specifying complex dependence on the past using intuitive and compact formulas, and building models of partially observable domains without specifying an underlying state space.

1. Introduction

The Markov assumption (MA) plays a key role in the definition of MDPs. It states that the next-state distribution following an action is independent of the history, given the current state. Yet, MA does not hold in many domains: one is likelier to be able to enter a restricted area if permission was obtained in the past; a car is likelier to skid if it rained in the past with no sunshine since; and even more so, if the rain was followed by below-zero temperature; a person is likelier to contract Covid flying if she has not contracted it in the past; or, a robot should be rewarded for delivering coffee only if its owner previously requested it. The first examples describe non-Markovian dynamics, and the last one, a non-Markovian reward, first studied in [1], and recently advocated as useful for reinforcement learning in robotics [2–5].

Often, MA is treated as a simple technical restriction because one can always modify the state description to make a non-Markovian model Markovian. For example, we can add indicator variables that record whether access was granted, coffee was requested, etc. Or, in POMDPs the belief-space MDP can be viewed as maintaining information about the entire history using an infinitely larger state space consisting of distributions over states.

This approach suffers from two problems: First, the agent's state space may be hard-coded. For example, a robot learning its environment may have a built-in state representation, related to its sensing abilities, set by a designer who was possibly unfamiliar with this environment. Second, even when the designer has the freedom to modify the state space, it is not clear a priori how to do this, when the dynamics is not clear. Do I need to remember everything that happened in the past? Do I need to remember that I was in a country afflicted by Malaria a few years ago? This turns out important if you want to donate blood, but otherwise may be unimportant. Do I need to remember that I picked up a cup? That I returned it to its place? And even if the designer is aware of the dynamics, it may be much simpler to specify non-Markovian dynamics or reward explicitly than to design the correct equivalent Markovian model which (as we show) can be exponentially, or even doubly exponentially, larger.

* Corresponding author.
E-mail addresses: brafman@cs.bgu.ac.il (R.I. Brafman), giuseppe.degiacomo@cs.ox.ac.uk (G. De Giacomo).

https://doi.org/10.1016/j.artint.2024.104113
Received 22 March 2023; Received in revised form 8 March 2024; Accepted 10 March 2024

Available online 21 March 2024

Past work has explored the specification of non-Markovian *reward* models using various temporal formalisms [1,6–8]. More recently, the direct use of automata for non-Markovian reward specification, suggested by [9,10] has become popular, and algorithms for learning non-Markovian rewards [11] and temporal-logic formula that model sets of traces [12] have been proposed. We build on many of these techniques. For example, the use of automata to transform non-Markovian models into Markovian models and the use of temporal logic to specify non-Markovian models were originally suggested in [1,6]. In the second part of this paper, we exploit the well-known relationship between automata and temporal logics, originally studied over infinite traces [13] and later extended to finite traces [14]. However, the introduction of a full non-Markovian decision model, first described in [19] and expanded on in this paper leads to a very interesting intermediate step between MDPs and POMDPs – the two central decision-theoretic models for sequential decision-making.

One well-known non-Markovian model is k-order MDPs, in which the next state distribution induced by an action depends on the last k states. However, k-order MDPs are impractical for large k, as their equivalent MDP is exponential in k. Moreover, they cannot express simple properties that depend on the arbitrary past, such as that the agent is carrying a key, or a credit card, picked up sometimes in the past, or any of the earlier examples.

To address these issues, we introduce and study *Regular Decision Process* (RDP), a non-Markovian decision model and language that can succinctly represent dependence on the arbitrary past. In an RDP, the next state and reward distribution can depend on the entire past history of actions and observations, much like in POMDPs. However, RDPs restrict this dependence to be *regular* in the language theoretic sense [15,16]. That is, the next state and reward distribution depend on which one of a finite number of memory elements, summarizing the relevant part of the current history, are true. It is easy to see that k-order MDPs are a special case of RDPs, and that RDPs are a special case of POMDPs.

Of particular interest are factored RDPs, in which the next state distribution and the reward function are conditioned on logical formulas, like in factored MDPs [17]. But while formulas in factored MDPs are propositional (or first-order) and refer to the current state only, formulas in RDPs are formulas in linear temporal logics of finite traces that are able to refer to the entire history. Concretely we can capture such properties using Linear Dynamic Logic over finite traces (LDL_f) [14], Linear Temporal Logic on finite traces (LTL_f) [14], Pure Past LTL (PPLTL), i.e., the linear time logic that refers only to the past [18], or even, directly, using regular expressions or finite state machines [15].

RDPs offer three potential benefits: First, they are easier to solve than POMDPs. In fact, their unfactored version is tractable. Second, they are more expressive than MDPs and can be used to approximate partially observable processes. Third, they do not require hypothesizing the existence of hidden variables or states – they are based on observable variables only. This latter property is promising from the learning perspective. The main contributions of this paper are to provide a comprehensive introduction to RDPs, first introduced in [19]. In particular, we study their properties and complexity, describe how to optimize them, and relate them to a number of similar existing dynamic models. The paper is structured as follows. First, we provide some background on related decision process models. In Section 3, we describe non-Markovian decision processes and define RDPs and their solution. In Section 4, we consider their relation to other decision models in the literature. In Section 5 we review some basic notions of temporal logics and use them to describe factored RDPs. We conclude in Section 6.

2. Background and related work

We review some existing decision process models, finite state machines, and temporal logics.

2.1. Decision process models

We review Markov Decision Processes (MDPs), k-order MDPs, POMDPs, and two special classes of POMDPs.

2.1.1. MDPs

A *Markov Decision Process (MDP)* is a tuple $\mathcal{M} = \langle S, A, Tr, R\rangle$ consisting of a set S of states, a set A of actions, a transition function $Tr : S \times A \to \Pi(S)$ that returns, for every state s and action a, a distribution over the next state,[1] and a reward function $R : S \times A \to \mathbb{R}$ that specifies the real-valued reward received by the agent when applying action a in state s. We sometimes abuse notation and treat Tr as a function from $S \times A \times S \to \mathbb{R}$ with the implicit assumption that this specifies a distribution.

The following parameters are often also specified as part of the model: the discount factor $0 < \gamma \leq 1$, a horizon h which specifies the number of actions/steps that the agent executes, and an initial state $s_0 \in S$. When we consider models with a fixed initial state, we sometimes write the MDP $\mathcal{M}$ as $\mathcal{M} = \langle S, A, Tr, R, \gamma, s_0\rangle$.

MDPs satisfy the Markov assumption because the effect of an action, as captured by the transition and the reward function, depends on the current state only, regardless of the more distant past.

Although we discuss state-based models, we are also interested in *factored* models. That is, models in which the state consists of value assignments to some set of variables. More specifically, in this paper states in S will often consist of truth assignments to a finite set $\mathbb{P}$ of primitive propositions. This allows for using a more compact, factored description of the reward and transition function using logical languages. To emphasize this, we write $\mathcal{M} = \langle \mathbb{P}, S, A, Tr, R, \gamma, s_0\rangle$ when referring to a factored MDP.

[1] $\Pi(X)$ denotes the space of probability distributions over a set X. $\pi(X)$ denotes a specific distribution over X.

A solution to an MDP is a *policy* ρ that maps histories of states and actions to actions. Because of the Markov assumption, we can restrict our attention to policies that only depend on the last state and the horizon, and in the case of infinite horizon, on the last state only. Hence, $\rho : S \rightarrow A$. The *value* of policy ρ at s, $v^{\rho}(s)$, is the expected sum of discounted rewards when starting at s and selecting actions based on ρ:

$$v_{\rho}(s) = \mathbb{E} \sum_{i=0}^{\infty} \gamma^{i} r_{i}$$

where r_i is a random variable whose value is the reward obtained in step i when executing policy ρ starting in state s.

Every infinite-horizon MDP has an *optimal* policy, ρ^*, that is stationary (i.e., depends on the current state only) deterministic, and maximizes the expected discounted sum of rewards for every starting state $s \in S$ [20]. Many methods for solving MDPs rely on MA, and their theoretical and practical complexity is strongly impacted by $|S||A|$.

2.1.2. *k-order MDP*

The MA implies that the state of an MDP summarizes all information that is required to predict the (stochastic) future. Often, the agent state is equated with its current observations, and the observation state may not satisfy the MA. One special, well-known case is that of k-order MDP in which the future depends on the last k states of the process. The components of a k-order MDP [21] are identical to those of an MDP, except that the transition function takes the form: $Tr : S^k \times A \rightarrow \Pi(S)$ and the reward function has the form: $R : S^k \times A \rightarrow \mathbb{R}$. While the above is the usual description, it is probably more accurate to view a k-order MDP as a model in which the state space is S^k, i.e., vectors of size k containing (original) states, and S is the observation space.

2.1.3. *Partially observable MDPs*

A partially observable MDP (POMDP) [22] is an MDP in which the decision maker does not have direct access to the current state. Instead, it receives an observation correlated with this state. Formally, a POMDP is a tuple $M = \langle S, A, Tr, R, O, \Omega, \gamma, b_0 \rangle$ where S, A, Tr, R, γ are as in MDPs, Ω is the set of possible observations, and $O : A \times S \rightarrow \Pi(\Omega)$ specifies the probability of observing $o \in \Omega$ if $a \in A$ was executed and led to state $s \in S$, denoted $o(a, s, o)$. Observations can be noisy, and the same observation may be possible given different state-action pairs. b_0 is the initial distribution over states.

While the MA holds w.r.t. the objective world dynamics, it does not hold w.r.t. the observations. That is, the information contained in the last observation received does not summarize the entire information available to the agent. Instead, the agent must consider the entire history of observations and actions. Thus, the agent's policy should depend on the entire history, and from this perspective, the process is non-Markovian. However, it is well known [22] that a POMDP can be solved by compiling it to a belief-space MDP – an MDP in which the states (called *belief states*) are distributions over S. The initial belief state is b_0, and given the current belief state, action, and observation, the next belief state can be computed using standard probabilistic inference. Thus, the non-Markovian view of a POMDP can be replaced by a Markovian model, albeit, over the uncountably large space of belief states.[2]

In a factored POMDPs [17], $M = \langle \mathbb{P}_S, S, A, Tr, R, \mathbb{P}_O, O, b_0 \rangle$, $\mathbb{P}_S$ and $\mathbb{P}_O$ are sets of propositions (potentially overlapping), S contains assignments to $\mathbb{P}_S$, and Ω contains assignments to subsets of $\mathbb{P}_O$. Transition, reward, and observation functions can be specified more succinctly by alluding to changes in the values of propositions as a function of the current state's properties using models such as dynamic Bayes-networks. RDDL [23] is a well-known format for such succinct representation.

2.1.4. *POMDP-lite*

POMDP-Lite [24] is a factored POMDP where $S = X \cdot \Theta$. X is a set of fully observable states, and Θ is a hidden parameter with a finite set of possible values $\theta_1, \ldots, \theta_n$. The transition function can be factored as $Tr((x, \theta), a, (x', \theta')) = Pr(x'|x, \theta, a) \cdot Pr(\theta'|x, \theta, a)$, where $Pr(\theta'|x, \theta, a)$ is restricted to be 0 or 1. That is, the hidden parameter value is either constant or changes deterministically. In this model, it is sufficient to maintain a belief state over Θ, and the authors show that this model is equivalent to a finite set of MDPs parameterized by θ.

A simple example of a POMDP-Lite model is the well-known Tiger domain, in which an agent has to select between two doors, behind one of which there is a tiger. The agent has three actions: two for opening a door and one for listening. Listening provides noisy information about the tiger's location, and opening a door ends the process with a reward that depends on the tiger's location. In this example, there is a single hidden parameter – the tiger's location. This parameter's value is unknown but constant. The set of states X consists of a single state. Imagine that after every two listening actions, the tiger changes its location. This is still a POMDP-Lite model because the value of the hidden parameter changes deterministically. We can further complicate this by assuming that the agent's ability to listen changes across time as a function of its current abilities. If the agent always knows what its listening ability is, then this is still a POMDP-Lite model with X being its possible "listening-ability" states.

2.1.5. *Deterministic labeled MDPs*

Deterministic Labeled MDPs (DLMDP) (also referred to as deterministic probabilistic automata) [25,26] are a model recently studied in the verification community. This model is essentially a special type of POMDP – as the name suggests, it is really an MDP with deterministic output. More specifically, it is deterministic in the following strong sense: The observation is a deterministic function of the state reached, and if the system can transition from state s to states s' and s'' on action a, then the output (i.e.,

[2] While this state space is uncountable, it enjoys special properties that are exploited by various solution algorithms.

observation) given states s' and s'' must be different. Moreover, the initial state is known (i.e., b_0 assigns probability 1 to some state s). If we know the initial state, then, by induction, we always know what state we are in based on the observations received so far. Formally (adapting the original notation to POMDPs), a DLMDP is a POMDP $M = \langle S, A, Tr, R, O, \Omega, \gamma, s_0 \rangle$, such that (a) $b_0(s_0) = 1$; (b) $\forall s \in S$ and $a \in A$ $\exists o \in \Omega$ such that $O(a, s, o) = 1$; (c) $\forall s, a, s', s''$ such that $s' \neq s''$, $Tr(s, a, s') > 0$ and $Tr(s, a, s'') > 0$, we have that $O(a, s', o) = 1 \rightarrow O(a, s'', o) = 0$.

2.1.6. *Predictive state representations*

Predictive State Representations (PSR) [27] are a model of dynamical systems that focuses on observable quantities and does not introduce nominal states into the picture. This is advantageous, especially in the context of learning, where we are only exposed to the actions performed and the observations obtained, and where the observer might not know what is the underlying space of system states. In fact, in some sense, such states may not objectively exist and are simply a tool for helping us predict future observations. As such, their utility in the model is questionable. To define a PSR, we first consider the infinite system-dynamics matrix $\mathcal{D}$. The rows of $\mathcal{D}$ correspond to possible action-observation histories – elements of $(A \cdot O)^*$. Its columns correspond to tests, which are also elements of $(A \cdot O)^*$. Given an ordering over the histories and tests, $\mathcal{D}_{il}$ is the probability that test $t_i = a^1 o^1 \cdots a^k o^k$ succeeds given history $h_l = a_1 o_1 \cdots a_j o_j$. That is, the probability that following history h, if actions $a^1, \ldots, a^k$ are performed, then observations $o^1, \ldots, o^k$ will be obtained.

The *linear dimension* of a dynamical system is defined as the rank of its system-dynamics matrix. When a system has a finite linear dimension, it can be represented by a finite set of linearly independent histories and a finite set of linearly independent tests. Given a set of linearly independent test columns, the tests they correspond to are called the *core tests* $Q = q_1, \ldots q_k$ and the state of the system given history h is given by the vector $p(Q|h) = [p(q_1|h), \ldots, p(q_k|h)]$. The initial state is $p(Q|\phi)$. Because Q is a basis for $\mathcal{D}$, for every test t, there exists a weight vector m_t such that $p(t|h) = p^T(Q|h) \cdot m_t$. One can show that using this, we can compute an update for each history, i.e., compute $p(Q|hao)$ from $p(Q|h)$ using a set of update parameters $\{m_{aoq} | a \in A, o \in O, q \in Q\}$ and $\{m_{ao} | a \in A, o \in O\}$. Finite-state POMDPs have a rank no greater than $|S|$ [27]. Consequently, any finite-state POMDP can be represented by a PSR with at most $|S|$ core tests.

2.2. *Finite state machines*

A finite-state machine M is defined as $M = \langle \Sigma, S, s_0, Tr, F \rangle$, where Σ is the input alphabet, S is a finite set of states, $s_0 \in S$ is the initial state, $Tr : S \times \Sigma \rightarrow S$ is the (deterministic) transition function, and $F \subseteq S$ are the accepting states. M is said to accept a word $\sigma_1 \cdots \sigma_k \in \Sigma^*$ iff $Tr(Tr(\cdots Tr(Tr(s_0, \sigma_1), \sigma_2) \cdots) \sigma_k) \in F$. That is, starting from the initial state, reading one character at a time, and transitioning according to Tr, we reach a state in F. It is well known from automata theory [15] that a language is accepted by an FSM *iff* it is a regular language.[3]

FSM has inputs but no outputs, except for the Boolean signal stating that the input sequence has reached a final state. *Finite-state transducers* extend FSMs with proper outputs: $M = \langle \Sigma, S, s_0, Tr, \Lambda, G \rangle$, where Λ is the output alphabet and G is the output function. Finite-state transducers can be either *Moore machines* or *Mealy machines*. In Moore machines, the output function depends on the current state only, i.e., $G : S \rightarrow \Lambda$. In Mealy machines, it depends on the previous state and input, i.e., $G : S \times \Sigma \rightarrow \Lambda$. Thus, Moore and Mealy machines can be viewed as mapping a sequence of inputs to a sequence of outputs: $\sigma_1 \cdots \sigma_k \rightarrow o_1 \cdots o_k$. Let $s_0, \cdots s_k$ be the sequence of states the machine moves through as $\sigma_1 \cdots \sigma_k$ is read. In a Moore machine $o_i = G(s_i)$ and in Mealy machine $o_i = G(s_{i-1}, \sigma_i)$. We say that two machines are equivalent if the mapping on input sequences they define is identical. It is well known that any Moore machine can be transformed into an equivalent Mealy machine and vice versa [28] and that this transformation is polynomial in the original machine's size. For this reason, we will refer generically to finite-state transducers, allowing for the use of Moore or Mealy machines, as convenient.

2.3. *Regular functions*

We define the concept of a regular function over sequences, which plays a key role in the definition of RDPs.

Definition 2.1. We say that a function $f : \Sigma^* \rightarrow \Lambda$ is *regular* if $range(f)$ is finite and for every $l \in \Lambda$, the language $f^{-1}(l)$ is regular.

That is, f is regular if we can "cover" Σ^*, the set of all possible finite words over alphabet Σ, by a finite set of mutually exclusive regular languages.

When f is regular, it can be represented by a transducer. This is formalized in the following lemma.

Lemma 2.2. *For every regular function $f : \Sigma^* \rightarrow \Lambda$ there exists a finite-state transducer $M = \langle \Sigma, S, s_0, Tr, \Lambda, G \rangle$ such that for every sequence $\sigma_1, \ldots, \sigma_k \in \Sigma^*$, we have that $G(Tr(\cdots Tr(Tr(s_0, \sigma_1), \sigma_2) \cdots, \sigma_k)) = f(\sigma_1, \ldots, \sigma_k)$.*

Proof. The intuitive idea is simple. The sequences mapping to each value in $range(f)$ form a regular language. Therefore, they can be identified by a finite-state machine. We can combine these FSMs into a single FSM by taking their product. Each state in this

[3] We assume familiarity with regular expressions and languages. See [28], for example.

product can be shown to have a component that is an accepting state of one of the original FSMs. We map this state into the value associated by f with the language accepted by this FSM.

More formally, let $\Lambda = range(f) = \{l_1, \ldots, l_m\}$, which we know to be finite. Let $M_l = \langle \Sigma, S_l, s_{0,l}, Tr_l, F_l \rangle$ be the FSM accepting $f^{-1}(l)$ for $l \in \Lambda$, which we know exists by regularity of f. Let $M_f = \times_{l \in \Lambda} M_l$ be the product FSM of all the M_l's. Recall that each $s \in S$ is a tuple of states $(s_1, \ldots, s_m) \in S_1 \times \cdots \times S_m$. Let S denote the states in M_f reachable from $s_0 = (s_{0,1}, \ldots, s_{0,m})$. Define $M = \langle \Sigma, S, s_0, Tr, \Lambda, G \rangle$, where Tr is the transition function of M_f and $G(s) = G(s_1, \ldots, s_m) = l_i$ iff $s_i \in F_i$. That is, $G(s)$ returns l_i iff s_i is an accepting state in M_{l_i}. We claim that for every sequence $\sigma_1, \ldots, \sigma_k \in \Sigma^*$, it holds that $G(Tr(\cdots Tr(Tr(s_0, \sigma_1), \sigma_2) \cdots, \sigma_k))$ is well defined and equals $f(\sigma_1, \ldots, \sigma_k)$. Consider any input sequence $\sigma_1, \ldots, \sigma_k \in \Sigma^*$. Let $f(\sigma_1, \ldots, \sigma_k) = l_i$. Let $s = (s_1, \ldots, s_m)$ be the state reached in M on input $\sigma_1, \ldots, \sigma_k$. By the definition of $M_{l_1}, \ldots, M_{l_m}$, only M_{l_i} accepts $\sigma_1, \ldots, \sigma_k$. Therefore, only $s_i \in F_i$. So G is well defined, and its value on $\sigma_1, \ldots, \sigma_k$ is l_i. $\square$

2.4. Non-Markovian reward functions

It has long been observed [1,8] 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 only if a request for coffee was made sometimes in the past.

To extend MDPs with non-Markovian rewards we need a language for specifying such rewards. [1,6] use 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 an 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*. [8] 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. In both cases, the authors also provide algorithms for efficiently converting such non-Markovian rewards to Markovian rewards over an extended state-space. For example, we can transform the non-Markovian reward in the "bring-coffee" example above into a Markovian reward by adding a new proposition that records whether a "bring-coffee" command was issued earlier, and updating it appropriately after each action.

Temporal logics allow us to specify diverse properties of traces. We discuss them in more depth in Section 5 and utilize them to specify RDPs. There is a well-known relation between properties captured by temporal logics (and their extensions to dynamic logics [29]) and automata. Given a formula in languages such as LTL_f and LDL_f, one can build an automaton that accepts only histories that satisfy the formula and vice versa. In fact, it is sometimes more natural and economical to directly specify properties of traces via such automata. For example, the Reward Machines formalism [30] specifies complex non-Markovian rewards by associating rewards with automaton states. One advantage of the automaton structure is that it can be naturally exploited during learning [10].

3. Non-Markovian decision processes and regular decision processes

We define the basic semantic model of a Non-Markovian Decision Process (NMDP) and the specialized class of Regular Decision Processes (RDPs).

3.1. NMDPs

An NMDP is identical to an MDP, except that the transition and reward functions depend on the *entire* history (trace). Formally:

Definition 3.1. A Non-Markovian Decision Process (NMDP) is a tuple $M = \langle O, A, tr, r \rangle$, where O is a set of observations and A is a set of actions; $tr : (A \times O)^* \times A \to \Pi(O) \cup \{\bot\}$ is the transition function, i.e., $tr((a_1, o_1, \ldots, a_k, o_k), a, o')$ is the probability of obtaining observation o' when executing action a given history $a_1, o_1, \ldots, a_k, o_k$; and r is the reward function: $r : (A \times O)^* \times A \to [r_{min}, r_{max}]$ or $r : (A \times O)^* \to [r_{min}, r_{max}]$ if the reward depends on the post-action observation, too.

Notice that it is possible that a is not applicable given certain traces, in which case tr is mapped to $\bot$.

In what follows, we will often assume that O encodes the last executed action, too. Hence, we will sometimes write $o_1, \ldots, o_k$ instead of $a_1, o_1, \ldots, a_k, o_k$, and $r : O^* \to [r_{min}, r_{max}]$, while sometimes it will be more convenient and clear to mention the action explicitly. Notice also that the empty sequence ϵ represents the initial state of the system.

A *policy* for an NMDP is a partial function $\rho : O^* \to \mathcal{A}$ such that ρ is defined on every sequence $w \in O^*$ reachable under ρ, where reachability under ρ is defined inductively as follows: ϵ is reachable; if $w \in O^*$ is reachable and $tr(w, \rho(w), o') > 0$ then $w \cdot o'$ is reachable.

The *value* of a trace $o_1, \ldots, o_n$ is its discounted sum of rewards:

$$v(a_1, o_1, \ldots, a_n, o_n) = \sum_{i=1}^{n} \gamma^n r((a_1, o_1, \ldots, a_{i-1}, o_{i-1}, a_i)$$

where $0 < \gamma < 1$ is the discount factor. Because we assume the reward value is lower and upper bounded, this discounted sum is always finite and bounded from above and below. For every finite horizon n, ρ and M, define a distribution over possible n-step traces, and $v_n(\rho)$ denotes the expected value of length n traces with respect to this distribution. We denote the *value of a policy* ρ by $v(\rho) = \liminf_{n\to\infty} v_n(\rho)$. Finally, ρ is an *optimal policy* for M if for every policy ρ', we have that $v(\rho) \geq v(\rho')$.

For MDPs with an infinite horizon, we know that there exists an optimal stationary policy, i.e., a policy $\rho : S \to \mathcal{A}$. Thus, in MDPs we can restrict attention to this finite set of policies. For NMDPs, it is clear that we cannot restrict our attention to stationary policies. In fact, it is not a priori clear whether an optimal policy exists. However, since the value of a policy is bounded, there exists some value $v^* = \sup_{\text{policy } \rho} v(\rho)$ (since every bounded set of real numbers contains a supremum).

3.2. RDPs

NMDPs can refer to the unbounded past in arbitrary ways, and may not be finitely representable, so one cannot specify or learn them, in general. Hence, we focus our attention on a class of NMDPs in which a finite description is used to capture the properties of traces. Specifically, we focus on NMDPs in which the transition and reward functions are regular.

Definition 3.2. A *Regular Decision Process* (RDP) is an NMDP $M = \langle O, A, tr, r\rangle$ such that tr and r are regular functions.

That is, every sequence of actions and observations belongs to one of a finite set of regular languages, and for any two sequences $\bar{ao}_1$ and $\bar{ao}_2$ belonging to the same language $tr(\bar{ao}_1, a) = tr(\bar{ao}_2, a)$, and similarly w.r.t. r.

Example 3.3. Lionel enjoys playing football (a.k.a. soccer). Lionel can choose between playing in a game, practicing, or drinking yerba mate. He gets a high reward of 10 for scoring in a game, a reward of 1 for drinking mate, and a reward of –2 for practicing.

When Lionel plays or practices he can get injured, in which case he gets a reward of –2, and in the next two time steps he gets a reward of –10 for playing and –5 for practicing. The rewards are additive, e.g., if he scores while he was injured, he will get $10 + (-10)$. We assume Lionel cannot both score and get injured in the same game.

Lionel's chances of scoring in a game depend on whether he drank mate or not in the past. If he never drank mate, he cannot score. Otherwise, his chances of scoring depend on whether he practiced (at least once) in the last 2 time steps. Injury does not affect his scoring abilities, and its likelihood increases if Lionel did not practice recently.

Formally, O = {scored (s), injured (i), null (n)}, A = {play (pl), practice (pr), mate (m)}. In defining tr below, we use $\star$ as shorthand notation for $(A \times O)^*$ and as a wild-card (within an AO pair), we use $\bar{a}$ to denote any action but a:

- $tr(*\,(m,*) * (pr,*)(*,*), play) = [(scored, 0.9), (injured, 0.01), (null, 0.09)]$ (had mate sometime, and practiced two time steps ago before play)
- $tr(*\,(pr,*)(m,*), play) = [(scored, 0.9), (injured, 0.01), (null, 0.09)]$ (as above, except that mate was drank just before the playing)
- $tr(*\,(m,*) * (pr,*), play) = [(scored, 0.9), (injured, 0.01), (null, 0.09)]$ (had mate sometime, and practiced one time step ago before play)
- $tr(*\,(m,*) * (\overline{pr},*)(\overline{pr},*), play) = [(scored, 0.5), (injured, 0.1), (null, 0.4)]$ (had mate sometime, and didn't practice the last two time steps)
- $tr(*\,(m,*)(\overline{pr},*), play) = [(scored, 0.2), (injured, 0.1), (null, 0.7)]$ (had mate then didn't practice)
- $tr(*\,(\overline{pr},*)(m,*), play) = [(scored, 0.3), (injured, 0.1), (null, 0.6)]$ (didn't practice then mate)
- $tr((\overline{m},*)^*, play) = [(scored, 0), (injured, 0.2), (null, 0.8)]$ (never had mate before play)
- $tr(*, practice) = [(injured, 0.1), (null, 0.9)]$
- $tr(*, mate) = [(null, 1)]$

The reward function $r : (A \times O)^* \to \mathbb{R}$ is:

- $r(*\,(*,s)) = 10$ (scored)
- $r(*\,(pr,*)) = -2$ (practiced)
- $r(*\,(m,*)) = 1$ (drank mate)
- $r(*\,(*,i)) = -2$ (was injured)
- $r(*\,(pl,i)(A \times O)(pl,*)) = -10$ (played after being injured recently)
- $r(*\,(pl,i) \cdot (pl,*)) = -10$ (played after being injured recently)
- $r(*\,(*,i)A \times O(pr,*)) = -5$ (practiced after being injured recently)
- $r(*\,(*,i)(pr,*)) = -5$ (practiced after being injured recently)

This example illustrates a number of important points: First, defining regular functions using regular expressions can be quite cumbersome. For this reason, it is often more convenient to specify them using some temporal logic, as was done for non-Markovian rewards. We discuss this technique in the context of RDPs in Section 5. Second, notice that RDPs have no explicit notion of state. The definition is solely based on the observable action-observation history. Of course, the regularity property reveals that there is a naturally derived notion of a hidden state – the state of the transducer that captures the regular function. Identifying these latent properties can make the system's dynamics much more understandable. We discuss this representation, next.

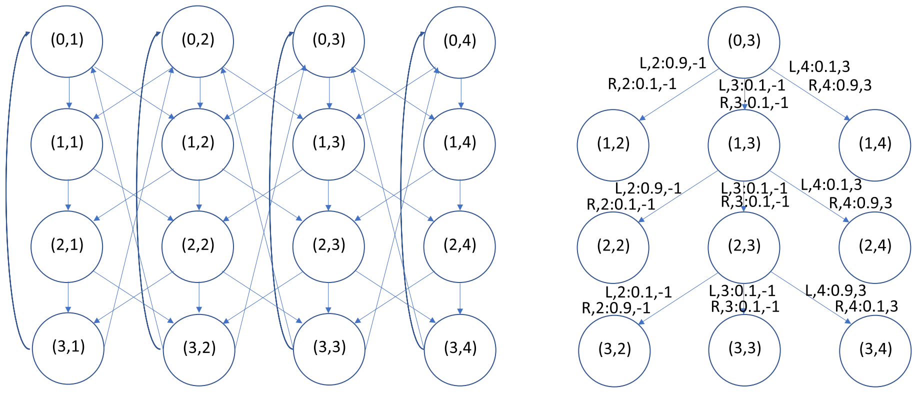


Fig. 1. Mealy Machine for Maze RDP. (a) Overview of states and transitions. (b) Specific transitions from states (0/1/2,3).

3.3. *FSM specification of an RDP*

Recently, FSM have been used to capture non-Markovian rewards in MDPs, [31–34,10]. This idea can be extended to RDPs. To see this, notice that Definition 3.2 associates each RDP with two regular functions: tr and r. By definition of these functions being regular, we know that they can be represented by a transducer $M = \langle S, \Sigma, Tr, \Lambda, G, s_0 \rangle$. Recall that one can use a Moore or a Mealy machine, and here it is more convenient and economical to choose a Mealy Machine. S is the set of transducer states, Tr is its (deterministic) transition function, and the alphabet Σ is $A \times O$. Thus, every sequence of actions and observations is mapped into some state $s \in S$, with the empty sequence being mapped to s_0. This Mealy Machine associates with every state s and action-observation pair (a, o), a pair (p, r), denoting the value of tr and r. Thus, $\Lambda = [0, 1] \times [R_{min}, R_{max}]$. G assigns to every state s and action observation pair (a, o) the probability of seeing o when we perform a and the reward r we would receive, given that the past action-observation sequence led us to s. Note that r depends only on the a component of the input (a, o), since we assume that the reward of an RDP depends on the current history and action, and not on the next observation.

In fact, it may be more natural and economical to use two Mealy Machines, one for tr and one for r (i.e., a reward machine), because the combined function operates on their product space, while the separate representation may allow for a more refined complexity analysis. For example, if we have two RDPs representing two tasks on the same dynamical model, we could amortize computations related to the dynamic model.

One reason the Mealy machine representation may be useful is that it suggests an approach for learning RDPs using Mealy Machine learning algorithms [35]. This direction is pursued in [36]. Because Mealy Machines are typically smaller than the equivalent Moore Machine, they are usually also easier to learn.

Example 3.4. Consider a linear maze with 4 cells $\{1, 2, 3, 4\}$, in which the agent has the actions *left (L)* and *right (R)*, following which it observes its current position. The left/right motion directions are relative to the agent's orientation. The agent starts in the leftmost position (1). The probability that it moves in the intended direction is 80%, and with probability 10% each, it stays in place or moves in the opposite direction. In case it moves into a wall, it remains in its current place. Initially, the agent is oriented up, but every two actions, its orientation flips 180^o. Thus, the effect of each action depends on how many actions modulo 4 have been executed so far. This property of the history is a regular property. Finally, there is a penalty of -1 for each move and a reward of $+3$ for arriving in position (4). For simplicity in this example, the reward is Markovian.

In Fig. 1(a), we see the structure of the Mealy Machine representing this RDP. For each possible observation, we need to distinguish between the number of actions performed so far, modulo 4. Each edge in this figure should actually be two edges, one for every possible action leading to the next state. In Fig. 1(b), we see the output function and edge labels for states $(0, 3), (1, 3), (2, 3)$. As noted above, each edge is really two edges, and therefore there are two labels for each edge. For example, from $(0, 3)$ on the action-observation pair $(L, 2)$ the next state is $(1, 2)$ and the output is $(0.9, -1)$, denoting the probability of this transition given action L. Notice that given the action observation $(L, 2)$, both the next state and the output are deterministic, as required. Similarly, another edge leading from $(0, 3)$ to $(1, 2)$ is labeled by $(R, 2)$. Its output is $(0.1, -1)$. Notice also that from $(2, 3)$ to $(3, 2)$ the output on $(L, 2)$ is $(0.1, -1)$ because the orientation is now flipped.

Finally, as we noted, one can use the less parsimonious Moore machine representation. In that case, we would have more states. A fragment of the relevant machine related to the $(0, 3)$ state is shown in Fig. 2, where the double edges are now clearer, as they lead to different states. The output is now associated with the states and not the edges.

3.4. *Solving RDPs*

RDPs are attractive because they provide a natural way of using the rich, yet intuitive language of regular expressions to specify a decision process in which transitions and rewards can depend on an unbounded history, unlike, for example, k-order MDPs. Of course, the ability to specify them is of little use if we cannot solve them.

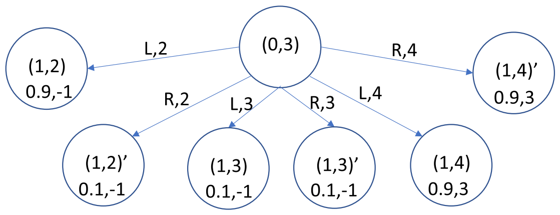


Fig. 2. A Fragment of a Corresponding Moore Machine.

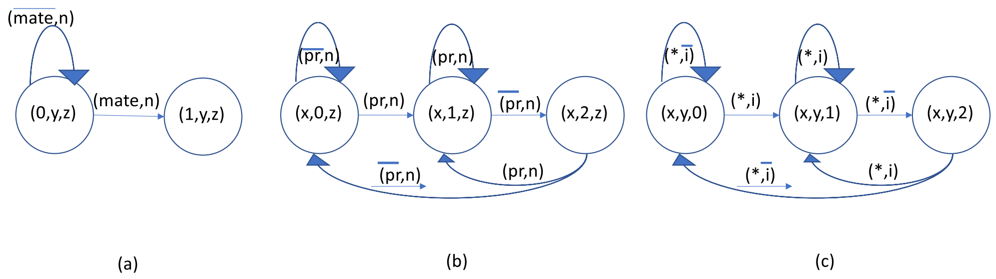


Fig. 3. The three automata making up the Mealy Machine for the Lionel Example.

A simple possible method for solving RDPs is to extend Monte-Carlo Tree-Search-based MDP solution algorithms [37] to RDPs. In principle, this requires little work and can be used for NMDPs in general. In MDPs, the children of the node depend on and are sampled based on the parent node only. In NMDPs, they depend on the entire path from the root node to the parent node, and the same algorithm can be applied by taking the information in this path into account. Applying this idea in general NMDPs can be costly because of the need to examine such long histories when expanding a node. Hence, node expansion and successor sampling become linear in depth instead of constant time. However, in RDPs, we can exploit the fact that the transition function is regular and can be encoded by an FSM. Thus, we can annotate each state in the search tree with the state of the relevant FSM, removing the dependence on history. POMCP [38], a well-known solution algorithm for POMDPs, does something similar. It "annotates" each node with its associated belief state, which is a summary of statistics of the entire history of actions and observations from the root. However, belief states are much costlier to update and consequently, POMCP's search algorithm maintains only a (possibly crude) approximation of belief states below the current node.

More generally, it is not difficult to see how an RDP can be transformed into an equivalent MDP. Consider an RDP $M = \langle O, A, tr, r\rangle$ and its Mealy Machine representation $M' = \langle Q, O, T, \Lambda, G, q_0\rangle$. For every sequence $\bar{o}$, tr and r depend on the state of M' after $\bar{o}$ and the next observation o (which contains the last action, too). We define an MDP with the same state space: $\mathcal{M} = \langle Q, A, Tr, R, q_0\rangle$ as follows: $Tr(q, a, q') = p$ and $R(q, a, q') = r$ if there exists an observation o containing action a such that $T(q, o) = q'$ and $G(q, o) = (p, r)$.

In words, the MDP has the same states as the RDP's Mealy Machine. Given a history leading to the Mealy Machine state q, and an action a, we seek those observations o consistent with a. We know that the transition function of the RDP depends on q, regardless of the particular history leading to it. So there is a probability and a reward associated with executing a in q and observing o. We associate these probabilities and rewards with the transition from q to $T(q, o)$ in the MDP, on action a. In principle, multiple a transitions can lead to the same next transducer state, in which case we would need to sum the probabilities.

The equivalence of the MDP and the RDP immediately follows from the fact that the RDP transition and reward functions depend only on the Mealy Machine state.

Example 3.5. Going back to Lionel, we can define an equivalent Mealy machine specification, which we can use to get an MDP. First, consider the states of the Mealy machine: we need to remember whether mate was drunk in the past, whether Lionel practiced during any one of the last two time steps, and whether he was injured in the last two time steps. This requires an automaton with 18 states, obtained by taking the product of three independent automata: one for mate (2 states), one for practices (3 states), and one for injuries (3 states). These automata are shown in Fig. 3. The Mealy machine then outputs for each state and edge the probability of making the relevant observation. The equivalent MDP has an identical state space and associates these probabilities and rewards with each transition. In Fig. 4 we show the transitions coming out of the initial state of the Mealy machine. The MDP is essentially identical.

3.5. Regular policies

A policy $\rho : O^* \to A$ is *regular* if ρ is a regular function. We know that this means that there exists a finite-state transducer that computes ρ. Such a transducer is more commonly known as a *finite state controller*. A by-product of the results above is:

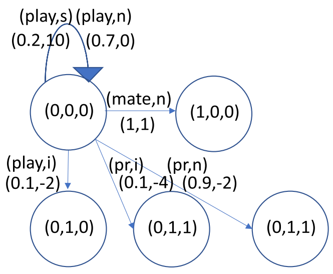


Fig. 4. Transitions from the initial state of the Mealy Machine/MDP.

Theorem 3.6. *Every RDP has a regular policy that is optimal.*

Proof. We know that every RDP is equivalent to an MDP over the states of the RDP's transducer. Every MDP has an optimal stationary Markov policy $\rho : S \to A$. Since S is a regular function of histories, so is ρ. $\square$

Thus, RDPs have optimal finite-state controllers, as opposed to general POMDPs, for example [39].

4. Relations to existing models

We now examine the relationship between RDPs and other models of dynamical systems. We rely strongly on the equivalence of an RDP to an MDP based on its equivalent Mealy Machine representation, and start with a few straightforward observations.

Observation 4.1. *Every k-order MDP is equivalent to an RDP whose associated FSM has size $O(|O|^k)$.*

A k-order MDP is an MDP whose transitions depend on the last k observations. Since we can build a regular expression for any finite sequence, we can represent such histories using regular expressions and build a corresponding RDP. At worst, the FSM of this RDP will have a state for every possible sequence of length k of observations, but depending on the transition function, fewer states may be needed.

Observation 4.2. *Every RDP is a POMDP-Lite model in which the initial belief state has support of size 1.*

A POMDP-Lite model is a product of an FSM and an MDP. An RDP has an identical structure, except that in an RDP, the initial state of the FSM is known, and in a POMDP-Lite model, it is unknown.

Observation 4.3. *Every RDP is a POMDP whose state space is equal to the size of the RDP's associated FSM.*

The MDP equivalent of an RDP, discussed in the previous section, is not a "real" MDP, in the sense that the state space and the observation space are not identical. It is a POMDP whose state space is the state space of the Mealy Machine representing the RDP, and its observations are the RDP's observations. This model can be viewed as a POMDP with one hidden variable that corresponds to the RDP's FSM's state, and its belief state always has support of 1.

One disadvantage of a POMDP is that its definition assumes the existence of a set of nominal states. This is a serious drawback when we consider the problem of learning such domains from observable data when no additional domain knowledge exists. This problem is well-known and has given rise to models such as predictive-state representations (PSR) [27] whose core idea is to work with the observations only.

Of course, one can take a history-based view of POMDPs, and represent them as a special class of NMDPs. However, general NMDPs are not a practical model because their specification requires explicating the dependence on history. In principle, one could specify that a certain transition probability exists for an action-observation pair, given that the history belongs to some undecidable language. RDPs focus on a particular variant of NMDPs that are finitely representable. Hence, RDPs could be a tractable approximation of POMDPs.

Indeed, in the discounted, infinite horizon case, with a bounded reward function, we can approximate a POMDP with a fixed initial belief state using an RDP (or an MDP) whose optimal policy is near-optimal for the POMDP by selecting a suitable effective horizon. (See also Theorem 5.6.) We say that policy ρ is *ϵ-optimal* if its value on the initial belief state is ϵ close to the optimal value, i.e., $|v^*(b_0) - v_\rho(b_0)| \le \epsilon$.

Lemma 4.4. *For every infinite horizon discounted reward POMDP M, and for every $\epsilon > 0$, there exists an RDP M_L such that the optimal policy for M_L is ϵ-optimal for M.*

Proof. It is trivially true that every finite-horizon POMDP is equivalent to an unfactored RDP because we care about finite histories only, and so the transition and reward functions are regular. For the unbounded case, the maximal contribution of the tail of the discounted sum can be made smaller than ϵ if we select a far enough effective horizon.

More formally, it is well known [20] that, letting v_t denote the value function of the policy that always applies the optimal first action in a t-step horizon MDP, we have that

$$||v^*(s) - v_t(s)|| \leq \frac{\gamma \cdot \delta}{1-\gamma}, \text{ such that } \delta = \sup_s ||v_t(s) - v_{t-1}(s)||.$$

Viewing a POMDP as a belief-state MDP, and since values are expectations over belief states, $\delta = \max_{s\in S} ||v_t(s) - v_{t-1}(s)||$. Keeping in mind that $\lim_{t\to\infty}\{||v_t(s) - v_{t-1}(s)||\} = 0$, we can select t far enough to ensure that $\frac{\gamma\cdot\delta}{1-\gamma} \leq \epsilon$. At that point, as above, we can use an RDP that is equivalent to the finite-horizon POMDP with the selected horizon t. A simpler off-line bound is obtained by bounding the maximal contribution of the tail of a geometric series $\gamma^i \cdot r_{max}$ that upper-bounds the possible gains from this tail by noting that the reward in each step is bounded by some value r_{max}. See also the footnote in the proof of Theorem 5.6. □

The above result can be trivially applied to belief-state MDPs, and is well-known. Specifically, we can consider all the finitely many belief states reachable within t steps from b_0,[4] and view them as states of a large, but still finite MDP. That is, we move to the belief-space MDP of the original POMDP but consider only its reachable fragment given a bounded horizon. This is essentially the same construction, only that in the RDP case, instead of inflating the state space, we remain with the same set of observations but will most likely need a more complex definition of the transition and reward functions.

Next, we consider DLMDPs and show their equivalence in the following sense:

Lemma 4.5. *For every RDP there exists a DLMDP such that $Pr(\bar{o}|\bar{a})$ and the reward associated with every transition is identical, and vice versa.*

Proof. We seek to prove the two models can induce the same distribution over observations given identical control signals. Intuitively, we saw that every RDP is equivalent to an MDP over an extended state space. If one looks closely, one sees that this MDP is actually more accurately a DLMDP – the observations in it are not the entire state, but they provide enough information to determine the state. For the other direction, we note that a DLMDP is a POMDP in which we always know what state we are in, given the last state and the current observation. This means that there is a finite-state machine that can be used to track the current state given the last state and the last observation. By annotating it with the probability of each transition (which is part of the POMDP's definition), we get a Mealy machine representation of an RDP. We give the formal proof, next.

Let $D = \langle O, A, tr, r\rangle$ be an RDP. Consider its Mealy Machine representation: $E = \langle S, A\times O, Tr, \Lambda, G, s_0\rangle$. Recall that G associates with each state $s \in S$ and every $(a, o) \in A\times O$, the probability of seeing o following a and the reward associated with this transition. We define the equivalent DLMDP to be $M = \langle S\times O, A, T, R, Obs, \Omega, (s_0, o)\rangle$. S, s_0, O and A are as in E. $\Omega = O$. Here the o part of the initial state (s_0, o) is an arbitrary observation in Ω. $Obs(a, o, (s, o)) = 1$. Given that it is a deterministic function of the state, we will abuse notation below and write $Obs(s, o) = o$. Suppose that $G(s, (a, o')) = (p, c)$. Then, $T((s, o), a, (s', o')) = p$ iff $Tr(s, (a, o')) = s'$. Finally, $R((s, o), a) = c$.

In words, we define a POMDP that satisfies the requirements of being a DLMDP. The states have the form (s, o), where s contains the needed information about the past, as captured in the Mealy Machine, and the last observation, o. The observation obtained in (s, o) is deterministically o, regardless of the action that led to (s, o). When we apply a in (s, o), we receive the reward associated with action a in state s of the Mealy Machine. (Recall that it does not depend on the following observation). We transition to the state (s', o'), where s' is determined deterministically by the transition function of the Mealy Machine and the probability of o' is determined by the p component of the output function of the Mealy Machine for state s given action (a, o'). The initial state of the POMDP is (s_0, o). The choice of o is not important as the first observation is obtained following some action.

To see that this is a DLMDP, first note that (a) there is a known initial state (s_0, o); (b) The observation function Obs is deterministic; and (c) suppose that $T((s, o), a, (s', o')) > 0$ and $T((s, o), a, (s'', o'')) > 0$. Suppose that $(s', o') \neq (s'', o'')$. We need to show that the $Obs(s', o') \neq Obs(s'', o'')$, i.e., that $o' \neq o''$. Suppose to the contrary that $o' = o''$. By definition of M, o' and o'' are the observations received, respectively, when applying a in s and reaching state s' in E, and when applying a in s and reaching s''. But if $o' = o''$ then $s' = s''$ because the transition function Tr of E is deterministic. Hence, $(s', o') = (s'', o'')$ in contradiction to our assumption.

Finally, we want to show the equivalence of D and M as formulated above. Consider an action sequence $a_1, a_2, \dots, a_n$. In the RDP: $Pr(o1, o2, \dots, o_n) = Pr(o1|a_1) \cdot Pr(o2|a_1, o_1, a_2) \cdots Pr(o_n|a_1, o_1, a_2, o_2, \dots, a_n)$. Denote by s_i the state obtained in the Mealy Machine following the sequence $a_1, o_1, a_2, o_2 \cdots, a_i, o_i$. Thus, this is equal to $Pr(o_1|s_0, a_1) \cdot Pr(o_2|s_1, a_2) \cdots Pr(o_n|s_{n-1}, a_n)$. But by definition $Pr(o_i|s_{i-1}, a_i)$ is exactly the probability of transitioning from (s_{i-1}, o) to (s_i, o_i) (regardless of what o is), in which case we observe o_i. So this is equivalent to the probability of passing through the states $(s_0, o), (s_1, o_1), \dots (s_n, o_n)$ given actions $a_1, a_2, \dots, a_n$. Since the choice of the s_i's here is deterministic given past actions and observations, there is no other sequence of states that we can pass when we apply $a_1, a_2, \dots, a_n$ and observe $o1, o2, \dots, o_n$. So the probabilities are equal.

Next, let $M = \langle Q, A, T, R, O, \Omega, s_0\rangle\rangle$ be a DLMDP. Recall that in a DLMDP, given the sequence of actions and observations (including the last observation), we know, by induction, what state we are in. We transform M to the form used in the other direction:

[4] We are assuming A is finite.

$M = \langle Q \times O, A, T', R, O', \Omega, (s_0, o)\rangle$ by recording the last observation in each state and modifying O' to be the projection to the observation part of the new state. We also need to modify the transition function as follows: Suppose we transition into (s', o'). As noted above, we know that s' is deterministic given the previous state and action and the current observation. Therefore, the probability of moving from (s, o) to (s', o') given a is 0 if s' is not the state we move to from s on a if we observed o'. Otherwise, it is the probability of observing o from that state (defined in the original observation function O). It is easy to see that these two models yield the same observations. Now, we can utilize the equivalence in the previous direction to construct an equivalent Mealy Machine. □

We see that for every DLMDP there is an RDP that has the same observed behavior and vice versa. But DLMDPs postulate an explicit underlying state space, essentially a POMDP, and then make a requirement that renders it fully observable, RDPs focus on the observation space only and characterize its non-Markovian property. In that sense, their perspective is similar to *Predictive State Representation* models (PSRs) [27].

It is interesting to compare RDPs with PSRs because PSRs were introduced explicitly to model dynamical systems without referring to a hidden state, and RDPs are also motivated by the desire to model rich dynamical systems without postulating some hidden state structure. [27] proved that a linear PSR can represent any POMDP using no more core tests than the number of underlying states of the POMDP. It also shows that the same holds for MDPs.[5] Indeed, for an RDP, the number of independent rows of the systems dynamics matrix is bounded by the number of states of the equivalent MDP. Moreover, the number of independent rows is at most the number of states, and every other row is equal to one of these independent rows. This follows because each history leads to a particular state, and given this state, no additional information about the past influences the future.

In [43,44], an example of an uncontrolled system is discussed that can be modeled by a PSR of dimension 3. This system cannot be modeled precisely by a hidden Markov model (uncontrolled POMDP) with a finite number of states, and consequently, it cannot be modeled by an RDP. It is approximable by a finite-state hidden Markov model [43], and we conjecture that also with an RDP.

To summarize the result, we introduce the following definition:

Definition 4.6. Given a class of models X, let $X_{O,A}$ denote the set of NMDPs with observations O and actions A representable by a *finite* model of class X with the same observation and actions set, O and A.

Theorem 4.7.

$$MDP_{O,A} \subseteq k{-}order MDP_{O,A} \subseteq RDP_{O,A}$$
$$RDP_{O,A} \subseteq POMDP{-}Lite_{O,A} \subseteq POMDP_{O,A} \subseteq PSR_{O,A}$$

Proof. In an MDP, $O = S$ and every MDP is a 1^{st}-order MDP and cannot express dependence on earlier observations. We know that every k-order MDP (for a fixed k) is an RDP whose next state depends on a restricted history, whereas an RDP can represent any dynamic system represented by the k-order MDP as well as systems in which an event in the unbounded past affects the system dynamics (e.g., I picked up a key some time ago). We know that an RDP is a special case of a POMDP-lite model. If we interpret the initial distribution as a choice of nature, then POMDP-lite is more general. It can describe an additional unobservable stochastic choice that cannot be captured by an RDP because in an RDP the stochastic effect of an action is observable. POMDP-Lite is a special case of a POMDP in which the hidden part is deterministic. Finally, we know that (a linear) PSR can represent a POMDP [27]. The result of [43,44] implies that a PSR of finite dimension requires a continuous state POMDP. □

We note that one may also say that $RDP_{O,A} \subsetneq POMDP_{O,A} \subsetneq PSR_{O,A}$. First, we know that every RDP can be viewed as an MDP over an extended, yet still finite state space S. And this MDP, in turn, can be viewed as a POMDP over the same action and observation sets, such that b_0 has support size one. The inclusion is strict because POMDPs can have an infinite number of reachable belief states, whereas the POMDP equivalent to any RDP has only a finite number of reachable belief states (because of the restriction on the initial belief state and the dynamics). The second relation follows from the fact that there are systems with finite linear dimension that cannot be modeled by a finite-state POMDP [44]. We note that this is not a clean comparison because we need to specify $|S|$ for POMDPs, whereas RDPs and PSR use O and A only. Interestingly, if we replace the stochastic transition function in RDPs and POMDPs with a non-deterministic one (obtaining the contingent planning model in the latter), the two models have equivalent expressive power [45].

Complexity-wise, a similar relation exists. There is a polynomial reduction from POMDPs to linear PSRs, so PSRs are at least as hard to solve as POMDPs [27,44]. POMDP optimization is *undecidable* for the infinite horizon case [41] and in PSPACE otherwise [40], while solving RDPs is polynomial in the size of the FSM.

[5] This may indicate that the number of core-tests and the linear dimension of a dynamic system may not be a good measure of the inherent complexity of a system, given that MDPs can be solved in time polynomial in $|S||A|$, whereas POMDPs are PSPACE-hard to solve exactly for the finite horizon case [40], undecidable in general [41] and are difficult to approximate [42].

5. Factored RDPs

We are especially interested in the factored version of RDPs in which observations are assignments to propositional variables, and actions and rewards are specified by referring to properties of sequences of such observations using formulas in an appropriate logic. Because we are interested in properties of finite sequences, the suitable logics are temporal logic on finite traces. We start this section with some background on these logics, use them to define factored RDPs, and discuss the complexity of their solution.

We note that there is much past work in the area of probabilistic model-checking and control on computing the probability that some given temporal formula will be satisfied [46], and on generating policies that ensure (or maximize the probability) that it will be satisfied [47]. Unlike them, we use temporal logics to describe the inherent dynamics and task structure of the system. This use of temporal logics can be viewed as extending their use beyond specifying non-Markovian rewards, as in [1,8,31,32,34,10], to specifying dynamics.

5.1. Background: temporal logics on finite traces

To declaratively express dependencies and properties of histories, we need to adopt a formalism that allows us to talk about finite traces. In recent years, several temporal logics with this ability have been studied. A first suitable logic is *Linear Temporal Logic on finite traces*, or LTL_f [14], which is the finite trace variant of *Linear Temporal Logic*, or LTL, [48], possibly the logic currently most used in Formal Methods [49]. LTL_f is as expressive as first-order logic on finite traces, or, equivalently, *star-free* regular expressions, which is a proper subclass of the regular expressions (RE). A more expressive variant of it is *Linear Dynamic Logic on finite traces*, or LDL_f [14], which extends LTL_f by allowing regular expressions in the temporal operator. LDL_f is inspired by *Propositional Dynamic Logic*, or PDL, [50–52], but it is interpreted over finite traces. LDL_f is as expressive as monadic second-order logic, or equivalently, full regular expressions, RE. There are two other logics that are of interest, they are *Pure Past* LTL, or PPLTL, and *Pure Past* LDL, or PPLDL, [18]. They correspond to the past variant of LTL_f and LDL_f respectively, and have exactly the expressive power of their future counterpart, but look at the history from the current instant towards the initial one, instead of from the initial instant to the current one. We will collectively refer to these logics as temporal logics on finite traces, and denote them generically with $\mathcal{TL}$.

The crucial result for all these logics is that for every formula φ there exists a FSM A_φ such that:

$$\tau \models \varphi \text{ iff } \tau \in \mathcal{L}(A_\varphi).$$

That is, a trace τ satisfies the formula φ if and only if it is accepted by the FSM A_φ. Constructing such a (minimal) FSM A_φ can be done in 2EXPTIME if φ is expressed in $\text{LDL}_f/\text{LTL}_f$ [14], and EXPTIME in φ is expressed in PPLDL/PPLTL [18]. The exponential advantage of the past logics w.r.t. the future one is due to a property of reverse languages, see [18]. In spite of the high worst-case complexity, in practice, computing the FSM of a formula is often easy, and there are tools that do it quite effectively (e.g., https://lydia-web-app.herokuapp.com/, http://ltlf2dfa.diag.uniroma1.it). Also the apparent advantage of past temporal logics w.r.t. future ones that incur a second exponential blow up due to determinization, most often does not manifest itself, as discussed in [53,54].

In this paper, for concreteness, we will focus on LDL_f, and specifically on a variant that also works on empty traces [32]. *However, much of this paper can be understood without familiarity with any temporal logic on finite traces. It will suffice to substitute "regular expressions" wherever "*$\text{LDL}_f/\text{LTL}_f$/PPLDL/PPLTL *formula" appears.* Observe that, RE themselves *are not convenient* for expressing temporal specifications: To negate a RE, one must generate the corresponding FSM, which can be exponential in the size of the RE, negate it, and generate its corresponding RE. One can conjoin two RE in polynomial time by constructing their nondeterministic automata (NFAs), generating their product NFA, and then moving to RE. Although this operation is polynomial, it is quite inconvenient. LDL_f was explicitly introduced in [14] to provide a more convenient specification language.

We start from a set $\mathbb{P}$ of propositions. The possible states or observations are truth assignments over $\mathbb{P}$. Thus, propositional formulas over $\mathbb{P}$ capture properties of states or observations. LDL_f formulas φ are built as follows:

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

tt stands for logical true; ϕ is a propositional formula over $\mathbb{P}$; ϱ denotes trace expressions, which are RE over propositional formulas ϕ with the addition of the test construct $\varphi?$ typical of PDL; $\langle\varrho\rangle\varphi$ states that, in the trace, from the current position, there exists a prefix satisfying the RE ϱ that reaches a position satisfying φ; tests are used to insert checks for satisfaction of additional LDL_f formulas in specified positions along the trace.

The semantics of LDL_f is given in terms of finite traces, i.e., finite sequences $\tau = \tau_1, \ldots, \tau_n$ of elements from the alphabet $2^{\mathbb{P}}$. In decision processes, executions are sequences of actions and states (or observations): $\langle a_1, s_1, \ldots, a_n, s_n\rangle$. We represent them as a trace τ by extending the set $\mathbb{P}$ to include one proposition p_a per action a, assigned true if a was the last action, and setting $\tau_i \doteq s_i \cup \{p_a \mid a = a_i\}$. In this way, τ_i denotes the pair (a_i, s_i) (with a dummy action a_1 for the initial state s_1). Henceforth, we assume this form, even if verbally referring to state-action sequences and sometimes representing the actions explicitly. We define $\tau(i) \doteq \tau_i$, $length(\tau) \doteq n$, and $\tau(i,j) \doteq \tau_i, \tau_{i+1}, \ldots, \tau_{j-1}$. When $j > length(\tau)$, $\tau(i,j) \doteq \tau(i, length(\tau)+1)$. Given a finite trace τ, an LDL_f formula φ, and a position i, we define when φ *is true* at step i, written $\tau, i \models \varphi$, by (mutual) induction, as follows:

- $\tau, i \models tt$;
- $\tau, i \models \neg\varphi$ if $\tau, i \not\models \varphi$;

- $\tau, i \models \varphi_1 \wedge \varphi_2$ if $\tau, i \models \varphi_1$ and $\tau, i \models \varphi_2$;
- $\tau, i \models \langle \varrho \rangle \varphi$ if there exists $i \leq j$ such that $\tau(i,j) \in \mathcal{L}(\varrho)$ and $\tau, j \models \varphi$, where the relation $\tau(i,j) \in \mathcal{L}(\varrho)$ is as follows:
  - $\tau(i,j) \in \mathcal{L}(\phi)$ if $j{=}i{+}1$, $i \leq \mathit{length}(\tau)$, and $\tau(i) \models \phi$ (ϕ propositional);
  - $\tau(i,j) \in \mathcal{L}(\varphi?)$ if $j = i$ and $\tau, i \models \varphi$;
  - $\tau(i,j) \in \mathcal{L}(\varrho_1{+}\varrho_2)$ if $\tau(i,j) \in \mathcal{L}(\varrho_1)$ or $\tau(i,j) \in \mathcal{L}(\varrho_2)$;
  - $\tau(i,j) \in \mathcal{L}(\varrho_1; \varrho_2)$ if there exists $k \in [i,j]$ such that $\tau(i,k) \in \mathcal{L}(\varrho_1)$ and $\tau(k,j) \in \mathcal{L}(\varrho_2)$;
  - $\tau(i,j) \in \mathcal{L}(\varrho^*)$ if $j = i$ or there exists k such that $\tau(i,k) \in \mathcal{L}(\varrho)$ and $\tau(k,j) \in \mathcal{L}(\varrho^*)$.

Note that if $i > \mathit{length}(\tau)$, 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 τ ***satisfies*** an LDL$_f$ formula φ, written $\tau \models \varphi$, if $\tau, 0 \models \varphi$.

It is convenient to use several abbreviations, including the usual propositional abbreviations, the abbreviation $\mathit{ff} \doteq \neg \mathit{tt}$ for logical false, and the abbreviation $[\varrho]\varphi \doteq \neg \langle \varrho \rangle \neg \varphi$ that states that from the current position, all prefixes satisfying the RE ϱ are such that they reach a position satisfying φ.

Observe that in LDL$_f$, propositional formulas ϕ occur only in trace expressions. In particular, if we want to express that ϕ holds in the current position, we have to write $\langle \phi \rangle \mathit{true}$, which indeed means that ϕ holds in the current position if the position is smaller or equal to that of the length of the trace, or trivializes to false if the position is greater than the length of the trace.

Among the propositional abbreviations, we have $\mathit{true} \doteq a \vee \neg a$ for $a \in \mathbb{P}$. But notice that $\langle \mathit{true} \rangle \mathit{tt}$ is equivalent to tt only in the positions within the trace, while it trivializes to false for the positions greater than the length of the trace. We can indeed define the abbreviation $\mathit{end} \doteq \neg \langle \mathit{true} \rangle \mathit{tt}$ which is true only when the trace is already ended, i.e., in those positions that are greater than the length of the trace.

In LDL$_f$ we can easily express LTL$_f$ by the following translation function $t(\cdot)$ defined by induction: $t(\phi) = \langle \phi \rangle \mathit{tt}$ (where ϕ is a propositional formula); $t(\neg \varphi) = \neg t(\varphi)$; $t(\varphi_1 \wedge \varphi_2) = t(\varphi_1) \wedge t(\varphi_2)$; $t(\bigcirc \varphi) = \langle \mathit{true} \rangle (t(\varphi) \wedge \neg \mathit{end})$; $t(\varphi_1 \mathcal{U} \varphi_2) = \langle (t(\varphi_1)?; \mathit{true})^* \rangle (t(\varphi_2) \wedge \neg \mathit{end})$. LTL$_f$ can also express any RE r with the corresponding formula $\langle r \rangle \mathit{end}$.

In the examples below, we use a convenient abbreviation $\mathit{atlast}(\varrho) \doteq \langle \mathit{true}^*; \varrho \rangle \mathit{end}$ to say that the sequence ϱ happened just before the end of the trace. For example, $\mathit{atlast}(\mathit{Rain})$ means that at the very end of the trace it rained.

Example 5.1. Here are some LDL$_f$ formulas and their intuitive semantics: $\varphi_1 = \langle \mathit{true}^*; \mathit{Rain}; \mathit{true}^* \rangle \mathit{end}$ – "along the trace it eventually rained", or if we want to describe the trace from its last position, we can say "it rained in the past". $\varphi_2 = \mathit{atlast}(\mathit{Rain}; \mathit{Rain}; \mathit{Rain})\mathit{end}$ – "it rained in all of the last 3 time positions of the traces" (e.g., days). $\varphi_2' = \mathit{atlast}(\mathit{Rain}) \vee \mathit{atlast}(\mathit{Rain}; \mathit{true}) \vee \mathit{atlast}(\mathit{Rain}; \mathit{true}; \mathit{true})$ – "it rained in one of the last 3 time positions of the trace". $\varphi_3 = \langle \mathit{true}^*; \mathit{Rain}; (\neg(\mathit{Tmp}{>}5))^* \rangle \mathit{end}$ – "towards the end of the trace it rained and the temperature was not above 5^{o}C until the end", or describing the trace from its last position "it rained in the past, and the temperature was not above 5^{o}C since". $\varphi_4 = \langle \mathit{true}^*; \mathit{Rain}; \mathit{Tmp}{<}0; (\neg(\mathit{Tmp}{>}2))^* \rangle \mathit{end}$ – again describing the trace from its last position "it rained in the past, then the temperature was below 0, and since, it was not above 2^{o}C".

5.2. Factored RDPs

We can now define factored RDPs.

Definition 5.2. A *factored* Regular Decision Process (RDP) is an NMDP $M_F = \langle \mathbb{P}_O, A, T, R \rangle$, where $\mathbb{P}_O$ is a set of propositional observation variables that implicitly define the set of possible observations, O, which is the set of truth assignments over $\mathbb{P}_O$. T is a *finite* set of quadruples of the form: $(\varphi, a, P, \pi(P))$, where φ is a $\mathcal{TL}$ formula over $\mathbb{P}_O$; $a \in A$, $P \subseteq \mathbb{P}_O$ and $\pi(P)$ is a joint-distribution over P describing its post-action distribution; R is a *finite* set of pairs of the form (φ, r), where φ is a $\mathcal{TL}$ formula over $\mathbb{P}_O$, and $r \in \mathbb{R}$ is a real-valued reward.

Moreover, if $\{(\varphi_i, a, P_i, \pi_i(P_i)) | i \in I_a\}$ are all quadruples for a, then the φ_i's must be *mutually exclusive*, i.e., $\varphi_i \wedge \varphi_j$ is inconsistent, for $i \neq j$.

Intuitively, $(\varphi, a, P, \pi(P)) \in T$ indicates that when the history satisfies φ then the application of action a impacts the propositions in P, only. And the distribution of their new values is given by $\pi(P)$. Because different formulae for a specific action are inconsistent, at most one such tuple applies to any history and action. Because these formulae need not be exhaustive, an a transition may be impossible given some histories. Notice that the empty history denotes the initial state, and it can be explicitly captured in $\mathcal{TL}$. For example, in LDL$_f$ we can use the formula $\mathit{end} \doteq [\mathit{true}]\mathit{ff}$). In addition, by convention, we assume that the initial valuation of all propositions in $\mathbb{P}_O$ is *false*. Or alternatively, when specifying transition from the initial state, we let $P = \mathbb{P}_O$.

The semantics of this specification is given by mapping it to an unfactored NMDP. We then show that this NMDP is, in fact, an RDP. Formally: let $M_F = \langle \mathbb{P}_O, A, T, R \rangle$ be a factored RDP specification. Define its induced unfactored NMDP $M = \langle O, A, tr, r \rangle$ as follows:

- $O = 2^{\mathbb{P}_O}$, i.e., the observations are truth-assignments over $\mathbb{P}_O$. Given this, for a $\mathcal{TL}$ formula φ defined over $\mathbb{P}_O$, we can determine whether φ is satisfied given the sequence of observations $o_1, \ldots, o_k$. By convention, $\phi \models \neg p$ for every $p \in \mathbb{P}_O$.
- $tr((o_1, ..., o_k), a)$ is defined as follows:
  1. If there exists a quadruple $(\varphi, a, P, \pi(P)) \in T$ such that $o_1, ..., o_k \models \varphi$ then

   (a) $tr((o_1,...,o_k),a,o') = \pi(o'|_P)$ if o_k and o' agree on all variables in $\mathbb{P} \setminus P$. Here, $o'|_P$ denotes the restriction of o' to the propositions in P.
   (b) $tr((o_1,...,o_k),a,o') = 0$ otherwise.
2. Otherwise: $tr((o_1,...,o_k),a,o) = \bot$ for every $o \in O$.

- $r(o_1,\ldots,o_k) = \sum_{(\varphi,r)\in R \wedge o_0,o_1,\ldots,o_k \models \varphi} r$.

That is, given the current trace and action a, if no quadruple has a condition φ satisfied by $o_1,...,o_k$, then no transition is possible on a (and we map it to $\bot$). Otherwise, let $(\varphi, a, P, \pi(P))$ be such a quadruple. By our assumptions, it is the only one. The next observation o' will assign propositions not in P the same value as in o_k, the previous observation. Under this condition, the probability of o' is equal to the probability that π assigns to the value of the P propositions in o'.[6] The reward associated with a history is the sum of rewards associated with reward formulas in R that it satisfies. As before, by definition, r is bounded above and below.

Example 5.3. Again, for concreteness, we use LDL_f as regular temporal logic $\mathcal{TL}$. When driving from A to B after it has rained, followed by temperature below zero and very low temperature since, there is a 0.1 probability of reaching B with some damage, and 0.1 probability of not reaching B at all (with some damage). Using φ_4 defined earlier, and A, B, d for at_A, at_B, *damaged*, respectively, we can write: $(\varphi_4 \wedge atlast(A \wedge \neg d), drive, \{A, B, d\}, \pi)$, where $\pi(\neg A \wedge B \wedge d) = 0.1, \pi(\neg A \wedge \neg B \wedge d) = 0.1, \pi(\neg A \wedge B \wedge \neg d) = 0.8$. If it has only rained and the temperature was not high since, then these probabilities drop to 0.01. We can write: $(\varphi_3 \wedge \neg\varphi_4 \wedge atlast(A \wedge \neg d), drive, \{A, B, d\}, \pi)$, where $\pi(\neg A \wedge B \wedge d) = 0.01, \pi(\neg A \wedge \neg B \wedge d) = 0.01, \pi(\neg A \wedge B \wedge \neg d) = 0.98$. To reward the robot for delivering coffee to Ann only if she requested it earlier, we can write $(\langle true^*; RqstAnn, (\neg DlvdAnn)^*; DlvdAnn\rangle end, 10)$.

Lemma 5.4. *The NMDP $M = \langle O, A, tr, r\rangle$ induced by a factored RDP is an RDP.*

Proof. We need to establish the existence of a regular function $f : (A \times O)^* \to Q$ such that if $f(\bar{o_1}) = f(\bar{o_2})$ then $tr(\bar{o_1}, a, o') = tr(\bar{o_2}, a, o')$ and $r(\bar{o_1} \cdot o') = r(\bar{o_2} \cdot o')$.

To do so, proceed as follows: Enumerate the quadruples in T as $(\varphi_1, a_1, P_1, \pi(P_1))$, …, $(\varphi_m, a_m, P_m, \pi(P_m))$, and the pairs in R as $(\varphi_{m+1}, r_{m+1}), \ldots, (\varphi_n, r_n)$. For each formula φ_i, build the corresponding FSM $M_i = \langle Q_i, 2^{\mathbb{P}_O}, \delta_i, F_i, q_0^i\rangle$ that accepts exactly those traces that satisfy φ_i. $O = 2^{\mathbb{P}_O}$, the set of all truth assignments to the propositions in $\mathbb{P}_O$, is M_i's alphabet; q_0^i is M_i's initial state, δ_i is its transition function; and F_i is the set of accepting/goal states. We now construct the product FSM $M = M_1 \times \cdots \times M_n$. Each sequence can be viewed as a word over the alphabet O, and therefore, it is mapped to a unique state in M. Any two sequences that map to the same state of M, satisfy, by construction, the same formulae φ_i. Therefore, their transitions and rewards are identical. We note that the product FSM defined by this construction is exponential in the number of formulae φ_i. □

Factored RDPs allow us to model complex non-Markovian processes using intuitive logical formulas describing the relevant properties of the history. As we soon show, these RDPs can be mapped to MDPs, and this process can be fully automated by using the tools mentioned above. Of course, the expressive power of RDPs is strongly tied to that of regular languages, and dependence on non-regular constructs cannot be expressed by RDPs. For example, it is not possible to specify a reward for a robot when the number of coffee cups served is equal precisely to the number of coffee requests made, if the number of requests cannot be a priori bounded. This requires the strength of a context-free language.

5.3. Solving factored RDPs

Given that every factored RDP is an RDP, factored RDPs, too, can be transformed into MDPs, given our earlier discussion of unfactored RDPs and the construction used in Lemma 5.4 that exploits the well-known relationship between $\mathcal{TL}$ logics and FSMs, for the various logics, such as LDL_f, LTL_f, PPLTL, and PPLDL. In fact, this solution technique is optimal w.r.t. worst-case computational complexity.

Theorem 5.5. *Given a factored RDP expressed in $\text{LDL}_f/\text{LTL}_f$ (resp. PPLDL/PPLTL), an optimal policy can be computed in 2EXPTIME (resp. EXPTIME); finding an optimal policy is 2EXPTIME-hard (resp. EXPTIME-hard).*

Proof. Membership comes from the construction in Lemma 5.4 above: the generation of the DFA from the formulas is in 2EXPTIME for $\text{LDL}_f/\text{LTL}_f$ (resp. in EXPTIME for PPLDL/PPLTL), their product is exponential in the number of quadruples, and computing a policy for an MDP is polynomial in $|S||A|$, where S is the resulting state space in Lemma 5.4. Hence the worst-case computational complexity is 2EXPTIME when we use $\text{LDL}_f/\text{LTL}_f$ for the specification of the quadruples (resp. EXPTIME, when we use PPLDL/PPLTL).

For the hardness, we resort to a reduction to FOND planning (under stochastic fairness) for $\text{LDL}_f/\text{LTL}_f$ (resp. PPLDL/PPLTL) goals. We observe that in FOND planning we are given a stochastic domain in which only the support for the transitions is known, but not the actual probability distribution. However, it has been recently shown that by setting any probability distribution compatible

[6] Other variants of this definition are possible. As long as what happens depends on properties of the trace leading to the current state that are expressible in $\mathcal{TL}$, the results that follow remain true.

with the given support and computing the optimal policy, we get a solution for the original FOND problem [55]. Hence solving an MDP with non-Markovian rewards specified in $\text{LDL}_f/\text{LTL}_f$ (resp. PPLDL/PPLTL) is at least as complex as solving FOND planning for $\text{LDL}_f/\text{LTL}_f$ (resp. PPLDL/PPLTL) goals. Hence it is 2EXPTIME-hard for $\text{LDL}_f/\text{LTL}_f$ [56,57] (resp. EXPTIME-hard for PPLDL/PPLTL [18]). Now we observe that MDPs with $\text{LDL}_f/\text{LTL}_f$ (resp. PPLDL/PPLTL) rewards are special cases of RDPs, hence we get the thesis.[7] □

Note that one exponential in the complexity of factored RDP optimization comes from the need to generate a factored representation of the FSM for each formula, which is worst-case exponential in the size of the formulas used. There is evidence that this transformation is fast in practice, and yields compact automata [53,54]. Moreover, as stated in the Theorem, when properties are described using pure past LTL/LDL, we can get rid of this exponential factor because, using these languages, one can refer to arbitrary events in the past using a bounded-size formula.

Observe that by Theorem 3.6, we know that RDPs admit optimal policies that are regular. When considering factored RDPs, we can also obtain *factored regular policies* ρ of the form $\{(\varphi_i, a_i)\}$ where φ_i is an LDL_f formula and a_i an action, such that for every trace $o_0, \ldots, o_k$ reachable given ρ, either no transition is possible after $o_0, \ldots, o_k$, or there exists exactly one $(\varphi_i, a_i) \in \rho$ such that $o_0, \ldots, o_k \models \varphi_i$. To see this, notice that the states of the Mealy Machine representing the RDP constructed according to Lemma 5.4 track the values of the various φ_i's. In every state of this Mealy Machine, either one formula φ_i is satisfied (and hence, this part of the policy can be captured by a pair (φ_i, a_i)) or it is a dead-end in which no action is possible.

5.4. Factored RDPs and POMDPs

Earlier, we noted the disadvantage of POMDPs' requirement for a nominal set of states and suggested RDPs as an approximation. We now repeat the construction of the approximating RDP in the factored case, which requires a little more effort.

Theorem 5.6. *For every infinite horizon discounted reward factored POMDP M, and for every $\epsilon > 0$, there exists a factored RDP M_L over $\mathbb{P}_O$, expressible in LDL_f (or PPLDL), such that the optimal policy for M_L is ϵ-optimal for M.*

Proof. First, we prove the following claim: *Every finite horizon factored POMDP M is equivalent to an RDP with variables $\mathbb{P} = \mathbb{P}_O$.* It is trivially true that every finite-horizon POMDP is equivalent to an unfactored RDP because the histories are all finite, and so the transition and reward functions are regular. Here we will show how to construct an equivalent factored RDP.

Starting from a factored POMDP $M = \langle \mathbb{P}_S, S, A, Tr, R, O, \mathbb{P}_O, b_0 \rangle$, let m be the horizon size. We define the factored RDP $M_L = \langle \mathbb{P}_O, A, tr_L, r_L, o_0 \rangle$. The observation set is the same as in the POMDP, and so are the observation propositions and the actions. To define tr_L, we first partition, for each action, all histories of length $\leq m$ into subsets $\{H_i^a | i \in I_a\}$ such that a changes the values of exactly the same proposition of the last observation when applied in each $h \in H_i^a$ – denote these propositions by P_i – and such that the marginal distribution over the P_i's values in $a(q)$ is the same for all $h \in H_i$ – denote this distribution by π_i. Some H_i may be singletons. We define $\mathcal{H} = \{(H_i^a, a, P_i, \pi_i) | i \in I_a, a \in A\}$.

Next, we construct a family of DFAs based on the history tree. These DFAs have identical states, input alphabet, transition function and initial state, and differ only in their accepting states. The DFA corresponds to the tree of all possible histories of length at most m, with o_0 added in front. Each state corresponds to a history reachable within at most m steps, starting with the empty history. The input alphabet is O. Given state(=history) h and an observation o, there is a transition from h on o to $h \cdot o$ if there is a positive probability of observing o given action a (encoded in o) and h in the POMDP.

For each $(H, a, P, \pi) \in \mathcal{H}$, H is a finite set of histories, and from basic automata theory, we know that there is a regular expression r_H that describes exactly the set of states of the above FSM that correspond to this set of histories. Given the expressive power of LDL_f, we can construct for each such set H an LDL_f formula $\varphi_H = \langle r_H \rangle end$ that is satisfied only given histories $h \in H$. Let $\Phi = \{(\varphi_H, a, P, \pi) : (H, a, P, \pi) \in \mathcal{H}\}$. Since, for every a, $\{H_i | i \in I_a\}$ partitions the set of histories, we know that no two different formulae $\varphi_H, \varphi_{H'}$ are satisfied by the same history. We can now define the quadruples for tr_L to be Φ. A similar method will let us generate the pairs of r_L.

Formally, to show that the RDP and the POMDP are equivalent we can proceed by induction to show that given any history h, the probability of a history $h \cdot o$ and the associated reward are identical in both models. This follows from our construction that simply represented the probability of the next observation (and reward) in relative terms (i.e., the changed propositions) and combined transitions that have an identical effect on the propositions. This holds true also for the histories of size 1, since we can describe any possible next observation by selecting the set of propositions to be the entire $\mathbb{P}_O$.

Let ρ be the optimal POMDP policy and v its value. Because of discounting, for every $\epsilon > 0$, ρ achieves value $\geq v - \epsilon$ within a finite number of steps m.[8] Thus, the optimal m-step policy for this POMDP has value $\geq v - \epsilon$. Since we constructed an equivalent RDP, its optimal policy has value $\geq v - \epsilon$. □

[7] The reader may wonder, whether the complexity is dominated by the reward specification as a $\text{LDL}_f/\text{LTL}_f$ (resp. PPLDL/PPLTL) formula, given that in the hardness proof above the dynamics is Markovian. However, this is not the case, since the resulting RDP that has a Markovian dynamics and a non-Markovian reward specification can be linearly transformed into an RDP with a non-Markovian dynamics and a Markovian reward. Let the non-Markovian reward specification be (φ, r), where φ is the original goal. We can define an RDP with a non-Markovian dynamics and a Markovian reward, by introducing a tuple $(\varphi, a, P_{new}, \pi(P_{new}) = 1)$ for any action a, where P_{new} is a new proposition, and changing the reward specification to $(atlast(P_{new}), r)$.

[8] Assuming rewards are bounded within $[0, 1]$ (the optimal policy is invariant under any affine scaling of rewards), we know that the rewards from step $m + 1$ onwards are bounded by $\gamma^m \sum_{i=1}^{\infty} \gamma^i = \frac{\gamma^m}{1-\gamma}$. We can then choose m such that $\frac{\gamma^m}{1-\gamma} \leq \epsilon$. Scaling to other ranges is immediate.

As we remarked w.r.t. Lemma 4.4, a similar approximation result for MDPs exists, where we move to a finite fragment of the belief-state MDP. In principle, we can define a factored representation of the states of this MDP, each of which is essentially a belief state, e.g., by enumerating them and using bit-vectors to encode these states. However, only the RDP-based approximation formulation preserves the original natural observation propositions, whereas, in the equivalent MDP, this structure will be corrupted.

It is interesting to note that the above proof did not utilize the full power of RDPs. It relied on the fact that finite languages are regular, and holds true for m-order MDP, as well. In the constructed RDP, every formula φ in the transition function specification (or equivalently, every state s of the transducer) corresponds to a finite set of histories. However, regular languages can be infinite. Hence, there may be more effective strategies for generating approximate RDP from POMDPs that utilize their full expressive power. This is an interesting challenge for future work.

6. Conclusion

We studied a new model for controlled dynamical systems. This model allows for compact and convenient specification of non-Markovian transitions and rewards, generalizes k-order MDPs, can ϵ-approximate POMDPs, yet makes no reference to hidden variables. This last property makes it a potential target model for learning dynamic systems without background knowledge. In future work, we hope to further explore methods for approximating POMDPs using RDPs, and to develop practical RDP learning algorithms. Finally, RDPs suggest that it may be worth studying additional models that restrict the nature of the hidden state of a POMDP for which better inference and learning algorithms may exist. Specifically, RDPs focus on domains in which the next observation is a stochastic function of a regular property of history, and more expressive variants could consider properties of history that are more expressive, such as context-free, etc.

CRediT authorship contribution statement

Ronen I. Brafman: Conceptualization, Formal analysis, Writing – original draft, Writing – review & editing. **Giuseppe De Giacomo:** Conceptualization, Formal analysis, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgements

This work is partially supported by the ERC Advanced Grant WhiteMech (No. 834228), by the Israel Ministry of Science and Technology (No. 3-15626), by the Israel Science Foundation (No. 1651/19), and by the Lynn and William Frankel Center for Computer Science at Ben-Gurion University of the Negev.

References

[1] F. Bacchus, C. Boutilier, A.J. Grove, Rewarding behaviors, in: AAAI, 1996.
[2] B. Lacerda, D. Parker, N. Hawes, Optimal and dynamic planning for Markov decision processes with co-safe LTL specifications, in: IROS, 2014.
[3] M.L. Littman, U. Topcu, J. Fu, C.L.I. Jr., M. Wen, J. MacGlashan, Environment-independent task specifications via GLTL, CoRR, arXiv:1704.04341 [abs], 2017.
[4] M. Hasanbeig, D. Kroening, A. Abate, Deep reinforcement learning with temporal logics, in: FORMATS, in: Lecture Notes in Computer Science, vol. 12288, Springer, 2020, pp. 1–22.
[5] C. Bradley, A. Pacheck, G.J. Stein, S. Castro, H. Kress-Gazit, N. Roy, Learning and planning for temporally extended tasks in unknown environments, in: ICRA, IEEE, 2021, pp. 4830–4836.
[6] F. Bacchus, C. Boutilier, A.J. Grove, Structured solution methods for non-Markovian decision processes, in: AAAI, 1997.
[7] S. Thiébaux, F. Kabanza, J.K. Slaney, Anytime state-based solution methods for decision processes with non-Markovian rewards, in: UAI, 2002.
[8] S. Thiébaux, C. Gretton, J.K. Slaney, D. Price, F. Kabanza, Decision-theoretic planning with non-Markovian rewards, J. Artif. Intell. Res. 25 (2006) 17–74.
[9] A. Camacho, O. Chen, S. Sanner, S.A. McIlraith, Decision-making with non-Markovian rewards: from LTL to automata-based reward shaping, in: RLDM, 2017, pp. 279–283.
[10] R. Toro Icarte, T.Q. Klassen, R.A. Valenzano, S.A. McIlraith, Reward machines: exploiting reward function structure in reinforcement learning, J. Artif. Intell. Res. 73 (2022) 173–208.
[11] M. Gaon, R.I. Brafman, Reinforcement learning with non-Markovian rewards, in: The Thirty-Fourth AAAI Conference on Artificial Intelligence, New York, NY, USA, February 7-12, 2020, AAAI Press, 2020, pp. 3980–3987.
[12] A. Camacho, S.A. McIlraith, Learning interpretable models expressed in linear temporal logic, in: J. Benton, N. Lipovetzky, E. Onaindia, D.E. Smith, S. Srivastava (Eds.), Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2019, Berkeley, CA, USA, July 11-15, 2019, AAAI Press, 2019, pp. 621–630, https://ojs.aaai.org/index.php/ICAPS/article/view/3529.
[13] M.Y. Vardi, An automata-theoretic approach to linear temporal logic, in: F. Moller, G.M. Birtwistle (Eds.), Logics for Concurrency - Structure Versus Automata, in: LNCS, vol. 1043, 1995.
[14] G. De Giacomo, M.Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: IJCAI'13, 2013.

[15] J.E. Hopcroft, J. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, 1979.
[16] M.O. Rabin, D. Scott, Finite automata and their decision problems, IBM J. Res. Dev. 3 (1959) 114–125.
[17] C. Boutilier, T. Dean, S. Hanks, Decision-theoretic planning: structural assumptions and computational leverage, J. Artif. Intell. Res. 11 (1999) 1–94.
[18] G. De Giacomo, A. Di Stasio, F. Fuggitti, S. Rubin, Pure-past linear temporal and dynamic logic on finite traces, in: IJCAI, 2020, pp. 4959–4965.
[19] R.I. Brafman, G. De Giacomo, Regular decision processes: a model for non-Markovian domains, in: IJCAI, 2019, pp. 5516–5522.
[20] M.L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, Wiley, 2005.
[21] W.K. Ching, M.K. Ng, Markov Chains: Models, Algorithms and Applications, International Series in Operations Research & Management Science, Springer, 2006, https://books.google.co.il/books?id=dr3fm2SAvr4C.
[22] K. Åström, Optimal control of Markov processes with incomplete state information, J. Math. Anal. Appl. 10 (1) (1965) 174–205, https://doi.org/10.1016/0022-247X(65)90154-X, https://www.sciencedirect.com/science/article/pii/0022247X6590154X.
[23] S. Sanner, Relational Dynamic Influence Diagram language (RDDL): Language description, 2010.
[24] M. Chen, E. Frazzoli, D. Hsu, W.S. Lee, POMDP-lite for robust robot planning under uncertainty, in: 2016 IEEE International Conference on Robotics and Automation, ICRA 2016, Stockholm, Sweden, May 16-21, 2016, 2016, pp. 5427–5433.
[25] H. Mao, Y. Chen, M. Jaeger, T.D. Nielsen, K.G. Larsen, B. Nielsen, Learning deterministic probabilistic automata from a model checking perspective, Mach. Learn. 105 (2) (2016) 255–299, https://doi.org/10.1007/s10994-016-5565-9.
[26] M. Tappler, B.K. Aichernig, G. Bacci, M. Eichlseder, K.G. Larsen, L^*-based learning of Markov decision processes, in: Formal Methods - the Next 30 Years - Third World Congress, FM 2019, Proceedings, Porto, Portugal, October 7-11, 2019, 2019, pp. 651–669.
[27] M.L. Littman, S.P. Singh, R.S. Sutton, Predictive representations of state, in: NIPS'01, MIT Press, 2001, pp. 1555–1561.
[28] J.E. Hopcroft, R. Motwani, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, international edition (2. ed), Addison-Wesley, 2003.
[29] M.Y. Vardi, From church and prior to PSL, in: 25 Years of Model Checking, in: Lecture Notes in Computer Science, vol. 5000, Springer, 2008, pp. 150–171.
[30] R. Toro Icarte, T.Q. Klassen, R.A. Valenzano, S.A. McIlraith, Using reward machines for high-level task specification and decomposition in reinforcement learning, in: ICML, in: Proceedings of Machine Learning Research, PMLR, vol. 80, 2018, pp. 2112–2121.
[31] A. Camacho, O. Chen, S. Sanner, S.A. McIlraith, Non-Markovian rewards expressed in LTL: guiding search via reward shaping, in: SoCS, 2017, pp. 159–160.
[32] R.I. Brafman, G. De Giacomo, F. Patrizi, LTLf/LDLf non-Markovian rewards, in: AAAI, 2018.
[33] R. Toro Icarte, T.Q. Klassen, R.A. Valenzano, S.A. McIlraith, Advice-based exploration in model-based reinforcement learning, in: Advances in Artificial Intelligence - 31st Canadian Conference on Artificial Intelligence, Canadian AI 2018, Proceedings, Toronto, ON, Canada, May 8-11, 2018, 2018, pp. 72–83.
[34] G. De Giacomo, L. Iocchi, M. Favorito, F. Patrizi, Foundations for restraining bolts: reinforcement learning with LTLf/LDLf restraining specifications, in: ICAPS, AAAI Press, 2019, pp. 128–136.
[35] S. Verwer, C.A. Hammerschmidt, flexfringe: a passive automaton learning package, in: 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17–22, 2017, IEEE Computer Society, 2017, pp. 638–642.
[36] E. Abadi, R.I. Brafman, Learning and solving regular decision processes, in: C. Bessiere (Ed.), Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20, International Joint Conferences on Artificial Intelligence Organization, 2020, pp. 1948–1954, Main track, https://doi.org/10.24963/ijcai.2020/270.
[37] L. Kocsis, C. Szepesvári, Bandit based Monte-Carlo planning, in: ECML, 2006.
[38] D. Silver, J. Veness, Monte-Carlo planning in large POMDPs, in: NIPS'10, 2010.
[39] E.A. Hansen, Solving POMDPs by searching in policy space, in: G.F. Cooper, S. Moral (Eds.), UAI '98: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, University of Wisconsin Business School, Madison, Wisconsin, USA, July 24-26, 1998, Morgan Kaufmann, 1998, pp. 211–219.
[40] C. Papadimitriou, J.N. Tsitsiklis, The complexity of Markov decision processes, Math. Oper. Res. 12 (3) (1987) 441–450.
[41] O. Madani, S. Hanks, A. Condon, On the undecidability of probabilistic planning and related stochastic optimization problems, Artif. Intell. 147 (1) (2003) 5–34, https://doi.org/10.1016/S0004-3702(02)00378-8.
[42] C. Lusena, J. Goldsmith, M. Mundhenk, Nonapproximability results for partially observable Markov decision processes, J. Artif. Intell. Res. 14 (2001) 83–103, https://doi.org/10.1613/JAIR.714.
[43] H. Jaeger, Discrete-time, discrete-valued observable operator models: a tutorial, Gmd report 42, German National Research Center for Information Technology, 1998.
[44] S. Singh, M.R. James, M.R. Rudary, Predictive state representations: a new theory for modeling dynamical systems, in: Proceedings of the Twentieth International Conference on Uncertainty in Artificial Intelligence, 2004, pp. 512–519.
[45] R.I. Brafman, G. De Giacomo, Planning for LTLf/LDLf goals in non-Markovian fully observable nondeterministic domains, in: IJCAI, 2019, pp. 1602–1608.
[46] M. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: verification of probabilistic real-time systems, in: G. Gopalakrishnan, S. Qadeer (Eds.), ICGI'11, in: LNCS, vol. 6806, Springer, 2011, pp. 585–591.
[47] X.C. Ding, S.L. Smith, C. Belta, D. Rus, Optimal control of Markov decision processes with linear temporal logic constraints, IEEE Trans. Autom. Control 59 (5) (2014) 1244–1257, https://doi.org/10.1109/TAC.2014.2298143.
[48] A. Pnueli, The temporal logic of programs, in: FOCS, 1977.
[49] C. Baier, J.-P. Katoen, K. Guldstrand Larsen, Principles of Model Checking, The MIT Press, 2008.
[50] M.J. Fischer, R.E. Ladner, Propositional dynamic logic of regular programs, J. Comput. Syst. Sci. 18 (1979).
[51] D. Harel, D. Kozen, J. Tiuryn, Dynamic Logic, MIT Press, 2000.
[52] M.Y. Vardi, The rise and fall of linear time logic, in: GandALF, 2011.
[53] D. Tabakov, M.Y. Vardi, Experimental evaluation of classical automata constructions, in: LPAR, 2005.
[54] S. Zhu, L.M. Tabajara, G. Pu, M.Y. Vardi, On the power of automata minimization in temporal synthesis, in: GandALF, in: EPTCS, vol. 346, 2021, pp. 117–134.
[55] B. Aminof, G. De Giacomo, S. Rubin, F. Zuleger, Beyond strong-cyclic: doing your best in stochastic environments, in: IJCAI, 2022, pp. 2525–2531.
[56] G. De Giacomo, S. Rubin, Automata-theoretic foundations of FOND planning for LTLf and LDL_f goals, in: IJCAI '18, 2018.
[57] B. Aminof, G. De Giacomo, S. Rubin, Stochastic fairness and language-theoretic fairness in planning in nondeterministic domains, in: ICAPS, AAAI Press, 2020, pp. 20–28.