Omega-Regular Decision Processes

Ernst Moritz Hahn[*1], **Mateo Perez**[*2], **Sven Schewe**[*3],
Fabio Somenzi[*2], **Ashutosh Trivedi**[*2], **Dominik Wojtczak**[*3]
[1]University of Twente, The Netherlands
[2]University of Colorado Boulder, USA
[3]University of Liverpool, UK
E.M.Hahn@utwente.nl, Mateo.Perez@colorado.edu, Sven.Schewe@liverpool.ac.uk,
Fabio@colorado.edu, Ashutosh.Trivedi@colorado.edu, D.Wojtczak@liv.ac.uk

Abstract

Regular decision processes (RDPs) are a subclass of non-Markovian decision processes where the transition and reward functions are guarded by some regular property of the past (a *lookback*). While RDPs enable intuitive and succinct representation of non-Markovian decision processes, their expressive power coincides with finite-state Markov decision processes (MDPs). We introduce omega-regular decision processes (ODPs) where the non-Markovian aspect of the transition and reward functions are extended to an ω-regular lookahead over the system evolution. Semantically, these lookaheads can be considered as *promises* made by the decision maker or the learning agent about her future behavior. In particular, we assume that if the promised lookaheads are not fulfilled, then the decision maker receives a payoff of $\perp$ (the least desirable payoff), overriding any rewards collected by the decision maker. We enable optimization and learning for ODPs under the discounted-reward objective by reducing them to lexicographic optimization and learning over finite MDPs. We present experimental results demonstrating the effectiveness of the proposed reduction.

Introduction

Markov decision processes (MDPs) are canonical models to express decision making under uncertainty, where the optimization objective is defined as a discounted sum of scalar rewards associated with various decisions. The optimal value and the optimal policies for MDPs can be computed efficiently via dynamic programming (Puterman 1994). When the environment is not explicitly known but can be sampled in repeated interactions, reinforcement learning (RL) (Sutton and Barto 2018) algorithms combine stochastic approximation with dynamic programming to compute optimal values and policies. RL, combined with deep learning (Goodfellow, Bengio, and Courville 2016), has emerged as a leading human-AI collaborative programming paradigm generating novel and creative solutions with "superhuman" efficiency (Silver et al. 2016; Wurman et al. 2022; Mirhoseini et al. 2020). A key shortcoming of this approach is the difficulty of translating designer's intent into a suitable reward signal. To help address this problem, we extend MDPs with a modeling primitive—called *promises*—that improves the communication between the agent and the programmer. We dub these processes *ω-regular decision processes* (ODPs).

Motivation. A key challenge in posing a decision problem as an MDP is to define a scalar *reward signal* that is Markovian (history-independent) on the state space. While some problems, such as reachability and safety, naturally lend themselves to a reward-based formulation, such an interface is often cumbersome and arguably error-prone. This difficulty has been well documented, especially within the RL literature, under different terms including *misaligned specification*, *specification gaming*, and *reward hacking* (Pan, Bhatia, and Steinhardt 2022; Amodei et al. 2016; Yuan et al. 2019; Skalse et al. 2022; Clark and Amodei 2016).

To overcome this challenge, automata and logic-based reward gadgets—such as reward machines, ω-regular languages, and LTL—have been proposed to extend the MDP in the context of planning (Baier and Katoen 2008) and, more recently, of RL (Icarte et al. 2018; Camacho et al. 2019; Sadigh et al. 2014; Hahn et al. 2019; Fu and Topcu 2014). In these works, an interpreter provides a reward for the actions of the decision maker by monitoring the action sequences with the help of the underlying reward gadget. While such reward interface is convenient from the programmer's perspective, it limits the agency of the decision maker in claiming rewards for her actions by making it opaque.

The formal study of non-Markovian MDPs in the planning setting was initiated by Brafman and De Giacomo (2019), who proposed regular decision processes (RDPs) as a tractable representation of non-Markovian MDPs. Abadi and Brafman (2021) further extended this work by combining Mealy machine learning with RL. In an RDP, the agent can choose a given action and collect its associated reward as long as the partial episode satisfies a certain regular property provided as the *guard* for that action. This modeling feature both permits and anticipates the agent to retain regular information about the past, enabling her to make optimal choices when selecting her actions. Augmenting MDPs with such *retrospective memory* offers a succinct and transparent modeling approach. However, adding memory as a regular language does not increase the expressive power of MDPs and RDPs

[*]These authors contributed equally.

where	LTL formula	type	reward
	$(\mathsf{G}\,\mathsf{F}\,\texttt{clean\_lab}) \land (\mathsf{G}\,\mathsf{F}\,\texttt{dirty\_lab})$	promise	0
	$\mathsf{F}\,\mathsf{G}\,\neg\texttt{initial\_location}$	promise	$-\xi$
	$\neg\texttt{dirty\_lab}\,\mathsf{S}\,\texttt{clean\_lab}$	guard	ρ
	$\texttt{decontamination}\,\mathsf{R}\,\neg\texttt{clean\_lab}$	promise	0
	$\texttt{true}$		$-f_1$
	$\texttt{true}$		$-f_2$

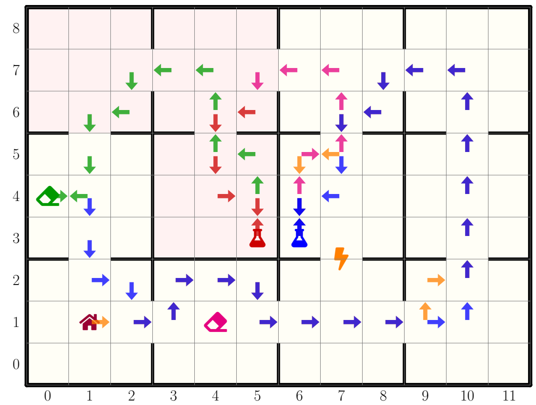

Figure 1: A grid-world model of a biological lab with clean and dirty areas. The strategy shown here is computed by RL based on the method proposed in this paper. The rewards satisfy $\xi > 0$ and $0 < f_1 < f_2 < \rho$. Promises and guards are specified in LTL (Pnueli 1977), with future and past operators. R denotes the *releases* operator: decontamination removes the constraint $\neg\texttt{clean\_lab}$. S denotes the *since* operator: the robot comes from the clean lab and has not been to the dirty lab since.

can be compiled into finite MDPs (Abadi and Brafman 2021) recovering the tractability of optimization and learning.

Prospective Memory. As a dual capability to the retrospective memory, we propose extending the RDP framework with the "prospective memory" (McDaniel and Einstein 2007) (also known as memory for intentions) to allow the agent to make promises about the future behavior and collect rewards based on this promise. We posit that such an abstraction will allow the agent to declare her intent to the environment and collect reward, and will result in more explainable and transparent behavior. This is the departure point for ω-regular decision processes, which we now introduce with the help of the following example. We note that while this example is little busy, it showcases multiple features of our framework.

Example 1 (Navigating a Biological Lab)**.** *Consider the grid world shown in Fig. 1, where a robot has to repeatedly visit two labs, one* `clean` *(blue) and one* `dirty` *(red). Whenever the robot passes through the* `dirty` *area—highlighted with a rose background—it has to visit a decontamination station (in one of the two cells marked with an eraser) before it can re-enter the* `clean` *lab. Every time the robot visits the* `dirty` *lab, it collects a reward if it just arrived from the* `clean` *lab.*

The two decontamination stations charge different fees. The cheaper one requires a detour from the shortest route. Both charge less than the robot earns by visiting the two labs. The clean lab has two doors. The one on the south side, however, is equipped with a "zapper" that has to be disabled on first crossing. If the robot manages to disable the zapper, it secures a shorter route and collects rewards more often; if it fails, it cannot complete its task. If the probability that the robot is put out of commission is sufficiently low, then a strategy that maximizes the expected cumulative reward will try to disable the zapper, while a strategy that maximizes the probability of carrying out the task will choose the longer, safer route. Let us assume the latter is desired. Finally, let us also assume that the robot should not re-enter its initial location more than a finite number of times.

Fig. 1 summarizes the specifications and details how they are expressed as rewards and promises. In this case, promises are associated to states; i.e., to all transitions emanating from the designated states. No lookbacks are necessary, though the promise made in the `dirty` *area could be turned into a guard on the entrance to the clean lab.*

The combination of ω-regular properties and rewards makes for a flexible and natural way to describe the objective of the decision maker. There may seem to be redundancy in the specification: why rewarding the robot for visiting the labs if it is already forced to visit them by the $\mathsf{G}\,\mathsf{F}$ *requirements? However, a proper combination of ω-regular and quantitative specifications may give strategies that simultaneously optimize short-term (discounted) reward and guarantee satisfaction of long-term goals (when such strategies exist). Without the ω-regular requirement, the robot of Fig. 1 would try its luck with the zapper. Without the reward collected on each visit to the dirty lab, the robot would only have ε-optimal strategies, which would postpone satisfaction of the ω-regular part of the specification to avoid the decontamination fees. Such postponement strategies are seldom practically satisfactory. Formulating the problem as an ω-regular decision process helps one prevent their occurrence. The strategy shown in Fig. 1 is computed using formal RL tool* MUNGOJERRIE *(Hahn et al. 2023a) based on the techniques presented in this paper.*

Contributions. This paper introduces ω-regular decision processes (ODPs) that generalize regular decision processes with prospective memory (promises) modeled as ω-regular lookaheads. We show decidability (Theorem 2) of the optimal discounted reward optimization problem for ODPs. In particular, we show that computing ε-optimal strategies is: 1) EXPTIME-hard when the lookaheads are given as universal co-Büchi automata (UCW) and 2) 2EXPTIME-hard when they are expressed in LTL.

A key construction of the paper is the translation of the lookaheads to a lexicographic optimization problem over MDPs. This construction creates a nondeterministic Büchi automaton (NBA) to test whether all promises made are almost surely fulfilled. This procedure involves a complementation procedure from UCAs to NBA. To be able to use this reduction for model checking or reinforcement learning, a critical requirement is to design an NBA that is *good-for-MDP* (GFM) (Hahn et al. 2020). We provide a rank based complementation construction to demonstrate that the resulting automata are GFM. We also show that leading rank-based complementation procedures all deliver GFM automata, enabling off-the-shelf complementation constructions to be used for OMDPs.

We have also implemented the proposed construction to remove ω-regular lookaheads from the MDPs. To demonstrate the experimental performance of our reduction, we present experiments on randomly generated examples. Some proof details have been omitted due to space limitations; the full version can be found on arXiv (Hahn et al. 2023b).

Preliminaries

Before we introduce ω-regular decision processes in the next section, we recall some basic definitions and notation in the simpler setting of Markov decision processes.

Markov Decision Processes

Let $\mathcal{D}(S)$ denote the set of all probability distributions over S. A Markov decision process $\mathcal{M}$ is a tuple (S, s_0, A, T, AP, L) where S is a finite set of states, $s_0 \in S$ is the initial state, A is a finite set of *actions*, $T\colon S \times A \to \mathcal{D}(S)$ is the probabilistic transition function, AP is the set of *atomic propositions* (observations), and $L\colon S \to 2^{AP}$ is the *labeling function*.

For any state $s \in S$, we let $A(s)$ denote the set of actions that can be selected in state s. An MDP is a Markov chain if $A(s)$ is singleton for all $s \in S$. For states $s, s' \in S$ and $a \in A(s)$, $T(s,a)(s')$ equals $\Pr(s'|s,a)$. A *run* of $\mathcal{M}$ is an ω-word $\langle s_0, a_1, s_1, \ldots\rangle \in S \times (A \times S)^\omega$ such that $\Pr(s_{i+1}|s_i, a_{i+1}){>}0$ for all $i \geq 0$. A finite run is a finite such sequence. We write $\mathit{Runs}^{\mathcal{M}}(\mathit{FRuns}^{\mathcal{M}})$ for the set of runs (finite runs) of the MDP $\mathcal{M}$ and $\mathit{Runs}^{\mathcal{M}}(s)(\mathit{FRuns}^{\mathcal{M}}(s))$ for the set of runs (finite runs) of the MDP $\mathcal{M}$ starting from the state s. We write $\mathit{last}(r)$ for the last state of finite run r.

We write $\Sigma \stackrel{\text{def}}{=} 2^{AP}$ for the alphabet of the set of labels. For a *run* $r = \langle s_0, a_1, s_1, \ldots\rangle$ we define the corresponding labeled run as $L(r) = \langle L(s_0), L(s_1), \ldots\rangle \in (\Sigma)^\omega$.

Strategies. A *strategy* in $\mathcal{M}$ is a function $\sigma\colon \mathit{FRuns} \to \mathcal{D}(A)$ such that$\mathit{supp}(\sigma(r)) \subseteq A(\mathit{last}(r))$, where $\mathit{supp}(d)$ denotes the support of the distribution d. A strategy σ is *pure* if $\sigma(r)$ is a point distribution for all runs $r \in \mathit{FRuns}^{\mathcal{M}}$ and is *mixed* if $\mathit{supp}(\sigma(r)) = A(\mathit{last}(r))$ for all runs $r \in \mathit{FRuns}^{\mathcal{M}}$. Let $\mathit{Runs}^{\mathcal{M}}_\sigma(s)$ denote the subset of runs $\mathit{Runs}^{\mathcal{M}}(s)$ that correspond to strategy σ with initial state s. Let $\Pi_{\mathcal{M}}$ be the set of all strategies. We say that σ is *stationary* if $\mathit{last}(r) = \mathit{last}(r')$ implies $\sigma(r) = \sigma(r')$ for all finite runs $r, r' \in \mathit{FRuns}^{\mathcal{M}}$. A stationary strategy can be given as a function $\sigma : S \to \mathcal{D}(A)$. A strategy is *positional* if it is both pure and stationary.

Probability Space. An MDP $\mathcal{M}$ under a strategy σ results in a Markov chain $\mathcal{M}_\sigma$. If σ is finite memory, then $\mathcal{M}_\sigma$ is a finite-state Markov chain. The behavior of $\mathcal{M}$ under a strategy σ from $s \in S$ is defined on the probability space $(\mathit{Runs}^{\mathcal{M}}_\sigma(s), \mathcal{F}_{\mathit{Runs}^{\mathcal{M}}_\sigma(s)}, \Pr^{\mathcal{M}}_\sigma(s))$ over the set of infinite runs of σ with starting state s. Given a random variable $f\colon \mathit{Runs}^{\mathcal{M}} \to \mathbb{R}$, we denote by $\mathbb{E}^{\mathcal{M}}_\sigma(s)\left\{f\right\}$ the expectation of f over the runs of $\mathcal{M}$ originating at s that follow σ.

Reward Machines. The learning objective over MDPs in RL is often expressed using a Markovian reward function, i.e., a function $\rho\colon S \times A \times S \to \mathbb{R}$ assigning utility to transitions. A *rewardful* MDP is a tuple $\mathcal{M} = (S, s_0, A, T, \rho)$ where S, s_0, A, and T are defined as for MDP, and ρ is a Markovian reward function. A rewardful MDP $\mathcal{M}$ under a strategy σ determines a sequence of random rewards $\rho(X_{i-1}, Y_i, X_i)_{i\geq 1}$, where X_i and Y_i are the random variables denoting the i-th state and action, respectively. For $\lambda \in [0,1[$, the *discounted reward* $\mathsf{EDisct}(\lambda)^{\mathcal{M}}_\sigma(s)$ from a state $s \in S$ under strategy σ is defined as

$$\lim_{N\to\infty} \mathbb{E}^{\mathcal{M}}_\sigma(s)\Big\{\sum_{1\leq i\leq N} \lambda^{i-1}\rho(X_{i-1}, Y_i, X_i)\Big\}. \qquad (1)$$

We define the optimal discounted reward $\mathsf{EDisct}^{\mathcal{M}}_*(s)$ for a state $s \in S$ as $\mathsf{EDisct}^{\mathcal{M}}_*(s) \stackrel{\text{def}}{=} \sup_{\sigma\in\Pi_{\mathcal{M}}} \mathsf{EDisct}^{\mathcal{M}}_\sigma(s)$. A strategy σ is discount-optimal if $\mathsf{EDisct}^{\mathcal{M}}_\sigma(s) = \mathsf{EDisct}^{\mathcal{M}}_*(s)$ for all $s{\in}S$. The optimal discounted cost can be computed in polynomial time (Puterman 1994).

Often, complex learning objectives cannot be expressed using Markovian reward signals. A recent trend is to resort to finite-state reward machines (Icarte et al. 2022). A reward machine is a tuple $\mathcal{R} = (\Sigma, U, u_0, \delta, \rho)$ where U is a finite set of states, $u_0{\in}U$ is the starting state, $\delta\colon U{\times}\Sigma \to 2^U$ is the transition function, and $\rho\colon U{\times}\Sigma{\times}U \to \mathbb{R}$ is the reward function. Given an MDP $\mathcal{M} = (S, s_0, A, T, AP, L)$ and a reward machine $\mathcal{R} = (2^{AP}, U, u_0, \delta, \rho)$, their product $\mathcal{M}{\times}\mathcal{R} = (S{\times}U, (s_0, u_0), (A{\times}U), T^\times, \rho^\times)$ is a rewardful MDP where the transition function $T^\times((s,u),(a,u'))((s',u'))$ equals $T(s,a)(s')$ if $u'{\in}\delta(u, L(s))$ and equals 0 otherwise. Moreover, the reward function $\rho^\times((s,u),(a,u'),(s',u'))$ equals $\rho(u, L(s), u')$ if $(u, L(s), u') \in \delta$ and is 0 otherwise. For discounted objectives, the optimal strategies of $\mathcal{M}{\times}\mathcal{R}$ are positional on $\mathcal{M}{\times}\mathcal{R}$, inducing finite memory strategies over $\mathcal{M}$ maximizing the learning objective given by $\mathcal{R}$.

Omega-Regular Languages

A deterministic finite state automaton (DFA) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$, where Σ is a finite *alphabet*, Q is a finite set of *states*, $\delta\colon Q \times \Sigma \to 2^Q$ is the *transition function*, and $F \subset Q$ is the set of *accepting (final) states*. A *run* r of $\mathcal{A}$ on $w = w_0 \ldots w_{n-1} \in \Sigma^*$ from an initial state $q_0 \in Q$ is a finite word $r_0, w_0, r_1, w_1, \ldots, r_n$ in $Q \times (\Sigma \times Q)^*$ such that $r_0 = q_0$ and, for $0 < i \leq n$, $r_i \in \delta(r_{i-1}, w_{i-1})$. We write $\mathit{last}(r)$ for the last state of the finite run r. A run r of $\mathcal{A}$ is *accepting* if $\mathit{last}(r) \in F$. The *language* $\mathcal{L}(\mathcal{A}, q)$ of $\mathcal{A}$ is the set of words in Σ^* with accepting runs in $\mathcal{A}$ from q.

ω-Automata. A (nondeterministic) *Büchi automaton* (NBA) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, \delta, \gamma)$, where Σ is a finite *alphabet*, Q is a finite set of *states*, $\delta\colon Q{\times}\Sigma \to 2^Q$ is the *transition function*, and $\gamma\colon Q \times \Sigma \to 2^Q$ with $\gamma(q,\sigma) \subseteq \delta(q,\sigma)$

for all $(q, \sigma) \in Q \times \Sigma$ are the *accepting transitions*. A *run* ρ of $\mathcal{A}$ on $w \in \Sigma^\omega$ from the initial state $q_0 \in Q$ is an ω-word $\rho_0, w_0, \rho_1, w_1, \dots$ in $(Q \times \Sigma)^\omega$ such that $\rho_0 = q_0$ and, for all $i > 0$, $\rho_i \in \delta(\rho_{i-1}, w_{i-1})$. We write $\inf(\rho)$ for the set of transitions that appear infinitely often in the run ρ. A run ρ of an NBA $\mathcal{A}$ is *accepting* if $\inf(\rho)$ contains a transition from γ. The *language* $\mathcal{L}(\mathcal{A}, q)$ of $\mathcal{A}$ is the subset of words in Σ^ω that have accepting runs in $\mathcal{A}$ from q. A language is *ω-regular* if it is accepted by a nondeterministic Büchi automaton.

A universal co-Büchi automaton (UCA) $\mathcal{A} = (\Sigma, Q, q_0, \delta, \gamma)$ is the dual of an NBA and its language can be defined using the notion of *rejecting runs*. We call a transition in γ *rejecting* and any runs with a transition in γ occurring infinitely often *rejecting* runs. The language $\mathcal{L}(\mathcal{A}, q)$ of a UCA $\mathcal{A}$ is the set of ω-words starting from q that do not have a rejecting run. A UCA therefore recognizes the complement of a structurally identical NBA.

Good-for-MDP Automata. Given an MDP $\mathcal{M}$ and a NBA automaton $\mathcal{A}$, the probabilistic model checking problem is to find a strategy that maximizes the probability of generating words in the language of $\mathcal{A}$. Automata-based tools provide an algorithm for probabilistic model checking when the NBA satisfies the so-called *good-for-MDP* property (Hahn et al. 2020). An NBA $\mathcal{A}$ is called *good-for-MDPs* if, for any MDP $\mathcal{M}$, controlling $\mathcal{M}$ to maximize the chance that its trace is in the language of $\mathcal{A}$ and controlling the syntactic product $\mathcal{M} \times \mathcal{A}$ (defined next) to maximize the chance of satisfying the Büchi objective are the same. In other words, for any MDP, the nondeterminism of $\mathcal{A}$ can be resolved on-the-fly.

Given an MDP $\mathcal{M} = (S, s_0, A, T, AP, L)$ and an (UCA or NBA) automaton $\mathcal{A} = (2^{AP}, Q, q_0, \delta, \gamma)$, their *product* $\mathcal{M} \times \mathcal{A} = (S \times Q, (s_0, q_0), A \times Q, T^\times, F^\times)$ is an MDP where the transition function $T^\times((s, q), (a, q'))((s', q'))$ equals $T(s, a)(s')$ if $(q, L(s, a, s'), q') \in \delta$ and it is 0 otherwise. The set of accepting transitions in the case of NBA or rejecting transitions in the case of UCA, $F^\times \subseteq (S \times Q) \times (A \times Q) \times (S \times Q)$, is defined by $((s, q), (a, q'), (s', q')) \in F^\times$ iff $(q, L(s, a, s'), q') \in F$ and $T(s, a)(s') > 0$. A strategy σ on the product induces a strategy σ' on the MDP with the same value, and vice versa. Note that for a stationary σ on the product, the strategy σ' on the MDP needs memory.

An *end-component* of an MDP $\mathcal{M}$ is a sub-MDP $\mathcal{M}'$ s.t. for every state pair (s, s') in $\mathcal{M}'$ there is a strategy to reach s' from s with positive probability. A maximal end-component is an end-component that is maximal under set-inclusion. An accepting/rejecting end-component is an end-component that contains an accepting/rejecting transition.

Omega-Regular Decision Processes

The Regular decision processes (RDPs) (Abadi and Brafman 2021) depart from the Markovian assumption of MDPs by allowing transitions and reward functions to be *guarded* (retrospective memory) by a regular property of the history. To build on this idea, we propose ω-regular decision processes (ODPs), where transitions and rewards are not only constrained by regular properties on the history but where the decision maker may also make *promises* (prospective memory) to limit their future choices in exchange for a better reward or evolution. ODPs offer a convenient framework for non-Markovian systems by allowing the decision maker to combine ω-regular objectives and scalar rewards.

For an automaton of any type, an *automaton schema* $\mathcal{A} = (\Sigma, Q, \delta, F)$ (for DFA) or $\mathcal{A} = (\Sigma, Q, \delta, \gamma)$ (for NBAs or UCAs) is defined as an automaton without an initial state. For an automaton schema $\mathcal{A} = (\Sigma, Q, \delta, \gamma)$ and a state $q \in Q$, we write $\mathcal{A}_q = (\Sigma, Q, q, \delta, \gamma)$ as the automaton with q as initial state and $\mathcal{L}(\mathcal{A}, q)$ for its language. We express various transition guards using a DFA schema (lookback automaton) and various promises using a UCA schema[1] (lookahead automaton).

Definition 1 (Omega-Regular Decision Processes)**.** *An ω-regular decision process (ODP) $\mathcal{M}$ is a tuple $(S, s_0, A, T, r, \mathcal{A}_a, \mathcal{A}_b, AP, L)$ where:*

- *S is a finite set of states,*
- *$s_0 \in S$ is the initial state,*
- *A is a finite set of actions,*
- *AP is the set of atomic propositions,*
- *$L : S \to 2^{AP}$ is the labeling function,*
- *$\mathcal{A}_b = (2^{AP}, Q_b, \delta_b, F_b)$ is a lookback DFA schema,*
- *$\mathcal{A}_a = (2^{AP}, Q_a, \delta_a, \gamma_a)$ is the lookahead UCA schema,*
- *$T : S \times Q_b \times A \times Q_a \to \mathcal{D}(S)$ is the transition function,*
- *and $r : S \times Q_b \times A \times Q_a \to \mathbb{R}$ is the reward function.*

An ODP with trivial lookahead $\mathcal{L}(\mathcal{A}_a, q) = \Sigma^\omega$, for every $q \in Q_a$, is a regular decision process (RDP)*. An ODP with trivial lookback $\mathcal{L}(\mathcal{A}_b, q) = \Sigma^*$, for every $q \in Q_b$, is a lookahead decision process (LDP). An ODP with trivial lookahead and lookback is simply an MDP. In these special cases, we will omit the trivial language from its description.*

A *run* $\langle s_0, (\beta_1, a_1, \alpha_1), s_1, (\beta_2, a_2, \alpha_2), \dots \rangle \in S \times ((Q_b \times A \times Q_a) \times S)^\omega$ of $\mathcal{M}$ is an ω-word such that $\Pr(s_{i+1} | s_i, (\beta_{i+1}, a_{i+1}, \alpha_{i+1})) > 0$ for all $i \geq 0$. A finite run is a finite such sequence. We say that a run $\langle s_0, (\beta_1, a_1, \alpha_1), s_1, (\beta_2, a_2, \alpha_2), \dots \rangle \in S \times ((Q_b \times A \times Q_a) \times S)^\omega$ is a *valid* run if for every $i \geq 1$ we have that $L(s_0)L(s_1) \cdots L(s_{i-1}) \in \mathcal{L}(\mathcal{A}_b, \beta_i)$ and $L(s_i)L(s_{i+1}) \cdots \in \mathcal{L}(\mathcal{A}_a, \alpha_i)$. The concepts of strategies, memory, and probability space are defined for the ODPs in an analogous manner to MDPs. We say that a strategy σ for an ODP is a *valid* strategy if the resulting runs are almost surely valid. Let $\overline{\Pi}_{\mathcal{M}}$ be the set of all valid strategies of $\mathcal{M}$.

[1]**Why UCAs?** We have opted for the use of UCAs, instead of NBAs, in our ODP framework due to the accumulation of promises during a run of an ODP. As new promises are made, previous promises must also be satisfied, leading to a straightforward operation on UCAs. However, this same operation on NBAs would result in alternating automata, adding an additional exponential blow-up to our construction. UCAs are becoming increasingly prevalent in both the formal methods (Finkbeiner and Schewe 2013; Filiot, Jin, and Raskin 2009; Dimitrova, Ghasemi, and Topcu 2018) and AI (Camacho et al. 2018; Camacho and McIlraith 2019) communities. They are often referred to as NBAs that recognize the complement language. It is worth noting that if an NBA $\mathcal{A}$ recognizes the models of an LTL or QPTL formula ϕ, or any other specification logic with negation, then $\mathcal{A}$, read as a UCA, recognizes $\neg\phi$ and vice versa. Therefore, the same automata translations can be applied to these specification languages.

The expected discounted reward $\mathsf{EDisct}(\lambda)^{\mathcal{M}}_{\sigma}(s)$ for a strategy in an ODP $\mathcal{M}$ is defined as in (1). We define the optimal discounted reward $\mathsf{EDisct}^{\mathcal{M}}_{*}(s)$ for a state $s \in S$ as $\mathsf{EDisct}^{\mathcal{M}}_{*}(s) \stackrel{\text{def}}{=} \sup_{\sigma\in\overline{\Pi}_{\mathcal{M}}} \mathsf{EDisct}^{\mathcal{M}}_{\sigma}(s)$. A strategy σ is discount-optimal if $\mathsf{EDisct}^{\mathcal{M}}_{\sigma}(s) = \mathsf{EDisct}^{\mathcal{M}}_{*}(s)$ for all $s\in S$. Given $\varepsilon > 0$, we say that a strategy σ is ε-optimal if $\mathsf{EDisct}^{\mathcal{M}}_{\sigma}(s) \geq \mathsf{EDisct}^{\mathcal{M}}_{*}(s) - \varepsilon$ for all $s\in S$. The key optimization problem for ODPs is to compute the optimal discounted reward and a discount-optimal strategy. However, such strategy may not always exist as shown next.

Example 2. *Consider an ODP where one can freely choose the next letter from the alphabet $\{a, b\}$ and have a reward of 1 for a and 0 for b. With each transition the lookback is trivial $\mathcal{L}(\mathcal{A}_b, q) = \Sigma^*$ and the lookahead is $\Sigma^*(b\Sigma^*)^\omega$ (infinitely many b's). While we cannot achieve the discounted reward of $\sum_{i=0}^{\infty} \lambda^i = \frac{1}{1-\lambda}$ with any valid strategy, we can get arbitrarily close to this value by, e.g., choosing a's until a reward $> \frac{1}{1-\lambda} - \varepsilon$ is collected for any given $\varepsilon>0$, and henceforth choose b's. While the optimal Büchi-discounted value is $\frac{1}{1-\lambda}$, no strategy can attain this value.*

Throughout the rest of this paper, we will focus on the problem of computing optimal discounted values and ε-optimal strategies for ODPs. Before we dive into the general problem, it is helpful to examine some important subclasses of ODPs.

Theorem 1 (Removing Lookbacks (Abadi and Brafman 2021))**.** *For any given RDP $\mathcal{M} = (S, s_0, A, T, r, \mathcal{A}_b)$, we can construct an MDP $\mathcal{N} = (S', s'_0, A', T', r')$ such that the optimal discounted value starting from s_0 in $\mathcal{M}$, denoted by $\mathsf{EDisct}^{\mathcal{M}}_{*}(s_0)$, is equal to the optimal discounted value starting from s'_0 in $\mathcal{N}$, denoted by $\mathsf{EDisct}^{\mathcal{N}}_{*}(s'_0)$. Moreover, a finite-memory optimal strategy for $\mathcal{M}$ can be computed from an optimal strategy for $\mathcal{N}$.*

Proof. Simulating the lookback automaton $\mathcal{A}_b$ is a straightforward process. Without loss of generality, we can assume that $(\mathcal{A}_b, p)$ is deterministic for all $p \in Q_b$. We can simulate $\mathcal{A}_b$ by computing, for each state $p \in Q_b$, the state $\alpha(p) \in Q_b$ that has been reached so far by $(\mathcal{A}_b, p)$ on the current prefix (if it exists; otherwise, $\alpha(p)$ is undefined). A transition of $(S, s_0, A, T, r, \mathcal{A}_b)$ with a lookback $r \in Q_b$ can be triggered whenever $F_b \cap \alpha(r) \neq \emptyset$. $\square$

Moving forward, we will assume that the ODP we are working with has a trivial lookback.

Complexity. It is easy to see that the optimization problem for ODPs is EXPTIME-hard, even for lookahead MDPs. This is due to the special case where the initial state of a lookahead MDP has no incoming transitions, and we can assign a payoff of 1 for the promise to satisfy a property given by a UCA and a 0 reward in all other cases. The problem then reduces to checking if the MDP can be controlled to create a word in the language of the UCA (or a model of the LTL formula) almost surely. If the specification can be satisfied almost surely, the expected reward will be 1, while it will be 0 otherwise. When this property is expressed in LTL, the complexity increases to 2EXPTIME-hard (Courcoubetis and Yannakakis 1995). Using the standard translation from LTL to NBAs and UCAs (e.g., (Somenzi and Bloem 2000; Babiak et al. 2012)), the complexity becomes EXPTIME-hard for the former.

Theorem 2 (Lower bounds)**.** *Finding an ε-optimal strategy for a lookahead decision process $\mathcal{M}_a = (S, s_0, A, T, r, \mathcal{A}_a)$ is EXPTIME-hard in the size of $\mathcal{A}_a$. If $\mathcal{A}_a$ is given as an LTL formula, the problem becomes 2EXPTIME-hard.*

Removing Lookaheads

The objective of this section is to establish a matching upper bound for Theorem 2. To meet the technical requirement of satisfying the ω-regular promises, we will translate them to good-for-MDP automata (Hahn et al. 2020). As the objectives are represented as universal co-Büchi automata, two operations are required: promise collection and translation to good-for-MDP NBAs. Promise collection is a simple operation for universal automata that does not impact the state space. However, translating an ordinary nondeterministic automaton to a good-for-MDP automaton, or even checking if an automaton has this property, can be a challenging task (Schewe, Tang, and Zhanabekova 2023). Complementation alone is a costly operation (Schewe 2009a).

We show that leading rank-based complementation procedures can be used to produce good-for-MDP (GFM) automata. Therefore, any standard implementation for automata complementation can be utilized. However, we suggest using a strongly limit-deterministic variant to avoid unnecessary nondeterminism, which is known (Hahn et al. 2020) to affect the efficiency of RL. Recall that an NBA is called *limit deterministic* if it is deterministic after seeing the first final transition. A limit deterministic automaton is *strongly* limit deterministic if it is also deterministic *before* taking the first final transition.

Definition 2. *An automaton is* strongly limit deterministic *if its state set Q can be partitioned into sets Q_1 and Q_2, such that $|\delta(q, \sigma) \cap Q_1| \leq 1$ for all $q \in Q_1$ and $\sigma \in \Sigma$ and $|\delta(q, \sigma)| \leq 1$ and $\delta(q, \sigma) \subseteq Q_2$ for all $q \in Q_2$ and $\sigma \in \Sigma$, and the image of γ is a subset of Q_2.*

From Ordinary to Collecting UCAs

We need to construct a GFM automaton that checks whether **all** promises made on the future development of the MDP are almost surely fulfilled. The first step is to transform the given UCA schema for testing individual promises into a UCA that checks whether all promises are fulfilled. When the promises are provided as states (or, indeed as sets of states) of a given UCA schema $\mathcal{A} = (\Sigma, Q, \delta, \gamma)$ and a fresh state $q'_0 \notin Q$ and $Q' = Q \cup \{q'_0\}$, we define the *collection automaton* $\mathcal{C} = (\Sigma \times Q, Q', q'_0, \delta', \gamma')$, whose inputs $\Sigma \times 2^Q$ contains the ordinary input letter and a fresh promise,

- $\delta'(q, (\sigma, q')) = \delta(q, \sigma)$ and $\gamma'(q, (\sigma, q')) = \gamma(q, \sigma)$ for all $q, q' \in Q$, that is, for states in Q, the promise is ignored, and
- $\gamma'(q'_0, (\sigma, q)) = \delta(q, \sigma)$ and $\delta'(q'_0, (\sigma, q)) = \{q'_0\} \cup \delta(q, \sigma)$, that is, from the fresh initial state q'_0, we have a non-final transition back to q'_0 as well as transitions that, broadly speaking, reflect the fresh promise $q \in Q$.

Note that promises can be restricted to be exactly or at most one state. The reason that the transitions from q_0' to other states are final is that this provides slightly smaller automata in the complementation (and determinisation) procedure we discuss in this section; as they can be taken only once on a run, it does not matter whether or not they are accepting, which can be exploited in a 'nondeterministic determinisation procedure' as in (Schewe 2009b).

Note that this automaton is easy to adjust to pledging acceptance from sets of states by using $\gamma'(q_0',(\sigma,S)) = \bigcup_{q\in S}\delta(q,\sigma)$ and $\delta'(q_0',(\sigma,S)) = \gamma'(q,(\sigma,S)) \cup \{q_0'\}$; the proofs in this section are easy to adjust to this case.

Theorem 3. *For a given UCA schema $\mathcal{A}$, the automaton $\mathcal{C}$ from above accepts a word $\varpi = (\sigma_0,q_0)(\sigma_1,q_1)(\sigma_2,q_2)\dots$ if, and only if, it satisfies all promises.*

From UCAs to (GFM) NBAs

Next, we consider a variation of the standard level ranking (Kupferman and Vardi 2001; Friedgut, Kupferman, and Vardi 2006; Schewe 2009a), which is producing a limit-deterministic automaton. This automaton is a syntactic subset in that it has the same states as (Schewe 2009a), but only a subset of its transitions. Besides being strongly limit-deterministic, we show that it retains the complement language and is good-for-MDPs. Our construction follows the intuitive data structure from (Schewe 2009a). It involves taking transitions away from the automaton resulting from the construction in (Schewe 2009a), so that one side of the language inclusions is obtained for free, while the other side is entailed by the simulation presented in the Appendix D of the full version (Hahn et al. 2023b) of this paper.

Construction. We call a level-ranking function $f: S \to \mathbb{N}$ from a finite set $S \subseteq Q$ of states S-tight if, for some $n \le |S|$, it maps S to $\{0,1,\dots,2n{-}1\}$ and onto $\{1,3,\dots,2n{-}1\}$. We write $\mathcal{T}_S$ for the set of S-tight level-ranking functions. We call $\mathsf{rank}(f) = \max\{f(q) \mid q \in S\}$ (the $2n{-}1$ from above) the *rank* of f.

Definition 3 (Rank-Based Construction)**.** *For a given ω-automaton $\mathcal{A} = (\Sigma,Q,I,\delta,\gamma)$ with $n = |Q|$ states, let $\mathcal{C} = (\Sigma,Q',\{I\},\delta',\gamma')$ denote the NBA where*

- *$Q' = Q_1 \cup Q_2$ with $Q_1 = 2^Q$ and $Q_2 = \{(S,O,f,i) \in 2^Q \times 2^Q \times \mathcal{T}_S \times \{0,2,\dots,2n-2\} \mid O \subseteq f^{-1}(i)\}$,*
- *$\delta' = \delta_1 \cup \delta_2 \cup \delta_3$ with*
  - *$\delta_1: Q_1 \times \Sigma \to 2^{Q_1}$ with $\delta_1(S,\sigma) = \{\delta(S,\sigma)\}$,*
  - *$\delta_2: Q_1 \times \Sigma \to 2^{Q_2}$ with $(S',O,f,i) \in \delta_2(S,\sigma)$ iff $S'{=}\delta(S,\sigma)$, $O{=}\emptyset$, and $i{=}0$,*
  - *$\delta_3: Q_2 \times \Sigma \to 2^{Q_2}$ with $(S',O',f',i') \in \delta_3\big((S,O,f,i),\sigma\big)$ iff the following holds: $S' = \delta(S,\sigma)$ and we define the auxiliary function $g\colon S' \to 2^{\{0,\dots,2n-1\}}$ with $g(q)$ equals*

    $$\big\{j \mid q \in \delta(f^{-1}(j),\sigma)\big\} \cup \big\{2\lfloor j/2\rfloor \mid q \in \gamma(f^{-1}(j),\sigma)\big\}$$

    f' is the S'-tight function with $f'(q) = \min\{g(q)\}$; if this function is not S'-tight, the transition blocks. Otherwise:
    (1) we set $O'' = \delta(O,\sigma) \cap {f'}^{-1}(i)$
    (2) if $O'' \neq \emptyset$, then $O' = O''$ and $i' = i$;
    (3) else $i'{=}(i{+}2) \mod (\mathsf{rank}(f'){+}1)$ and $O'{=}{f'}^{-1}(i')$
- *γ' contains the transitions of δ_3 from case (3) (the breakpoints) as well as transitions from $\{\emptyset\}$.*

Theorem 4 (Schewe (2009a))**.** *Given an NBA $\mathcal{A}$, the NBA $\mathcal{C}$ from Definition 3 recognizes a subset of the complement of the language of $\mathcal{A}$. i.e. $\mathcal{L}(\mathcal{C}) \subseteq \Sigma^\omega \setminus \mathcal{L}(\mathcal{A})$.*

Corollary 1. *Given a UCA $\mathcal{A}$, the NBA $\mathcal{C}$ from Definition 3 recognizes a subset of the language of $\mathcal{A}$, i.e. $\mathcal{L}(\mathcal{C}) \subseteq \mathcal{L}(\mathcal{A})$.*

Showing inclusion in the other direction (and thus language equivalence) can be done in two ways. One way is to re-visit the similar proof from the complementation construction from (Schewe 2009a). It revolves around guessing the correct level ranking once it is henceforth tight, and this guess, and its corresponding run, is still possible. However, as we need to establish that the resulting NBA $\mathcal{C}$ is good-for-MDPs, we take a different approach: we start from determinising the UCA $\mathcal{A}$ into a deterministic Streett automaton $\mathcal{S}$, using the standard determinisation from nondeterministic Büchi to deterministic Rabin automata (Schewe 2009b). It is then easy to see how an accepting run of $\mathcal{S}$ on a word can be simulated. The proof details are provided in the full version.

Theorem 5. *For a given UCA $\mathcal{A}$, the NBA $\mathcal{C}$ from Definition 3 is a language equivalent good-for-MDPs NBA.* □

Noting that the construction in Definition 3 is a language equivalent syntactic subset of (Schewe 2009a), which in turn is a language equivalent syntactic subset for older constructions (Kupferman and Vardi 2001; Friedgut, Kupferman, and Vardi 2006; Schewe 2009a), we obtain that the classic rank-based complementation algorithms result in GFM automata.

Corollary 2. *Given an NBA $\mathcal{A}$, the rank-based complementation algorithms from (Kupferman and Vardi 2001; Friedgut, Kupferman, and Vardi 2006; Schewe 2009a) provide good-for-MDP automata.* □

Appendix E of the full version (Hahn et al. 2023b) of this paper provides optimizations for this construction, showing in particular that (1) δ_2 can be restricted to map all states to odd ranks and that (2) the state q_0' from the collection automaton can always be chosen to be the sole state with maximal rank. Further, we argue that safety and reachability objectives lead to subset and breakpoint constructions, respectively.

Putting It All Together

Combining the selection of promises and their efficient representation as a GFM automaton, we have reduced our problem to a lexicographic optimization problem with an ω-regular and discounted reward objective, for which one can use model-checking and reinforcement learning approaches (Bozkurt, Wang, and Pajic 2021; Hahn et al. 2023c).

Theorem 6. *The problem of finding (near) optimal control for a lookahead decision process $\mathcal{M}_a = (S,s_0,A,T,r,\mathcal{A}_a)$ can be done in time polynomial in $\mathcal{M}$ and is EXPTIME-complete in the size of $\mathcal{A}_a$, and 2EXPTIME-complete in the size of an LTL formula describing $\mathcal{A}_a$.* □

	orig	compl	prune	lumpd	lang	lumpa	time
mean	4.09	48,367.45	3,748.94	23.80	7.77	7.03	0.40
stdev	2.91	760,963.32	129,045.67	211.58	10.39	8.70	6.39
max	34.00	25,107,909.00	9,152,588.00	9,958.00	327.00	320.00	269.61

Table 1: Statistics for randomly generated examples. *orig*: Number of states of the automaton generated by `ltl2tgba`, *compl*: Number of states of the complement, *prune*: Number of states after removing states with empty language, *lumpd*: Number of states after applying strong-bisimulation lumping in the final part of the automaton, *lang*: Number of states after we identify language-equivalent states in the final part and redirect transitions from the initial part to a representative for each language, *lumpa*: Number of states after applying strong bisimulation lumping for all states of the automaton, *time*: total time in seconds.

formula	orig	compl	prune	lumpd	lang	lumpa	time
`Fd U ((a <-> Gd) & (c <-> Fb))`	14	25,107,909	16,585	2,120	115	60	269.61
`((c xor Fd) R F(b & c)) W Xd`	13	20,484,339	59,150	1,005	30	13	127.77
`F((a W (1 U (d xor Xd))) R (a W c))`	10	19,317,020	18,540	103	40	29	167.43
`X(1 U a) R F(!Gb & (c W a))`	11	18,492,964	294,249	502	32	15	111.25
`G(Xa xor (G(Gc xor Ga) M Xd))`	14	18,129,540	9,152,588	909	80	73	112.71
`!G(a & c) | X!Xa`	2	4	2	2	2	2	0.00
`XG(Gd U (!a & (c M Ga)))`	1	2	2	2	2	2	0.00
`!(b M c) -> (c & X!b)`	3	6	3	3	3	3	0.00
`(Ga -> b) U c`	4	8	6	6	6	6	0.01
`(!c R Fb) U (Gd <-> GFb)`	10	232,094	70,513	6,481	60	15	11.76

Table 2: Example formulas. For the legend, see Table 1.

Experimental Results

Our construction effectively reduces the optimization and RL problem for ODPs to lexicographic optimization/RL over MDPs. For our experiments, we focus on showing that what could be a computational bottleneck (the size of resulting Büchi automaton) is not a showstopper. Once the automaton is produced, the scalability of our approach is similar to that of the lexicographic planning and RL algorithms (Hahn et al. 2023c). For instance, we combined our construction with the lexicographic ω-regular and discounted objectives RL algorithm introduced in (Hahn et al. 2023c) to compute optimal policies shown in Figure 1. It took 20 mins on Intel $i7{-}8750H$ processor.

Efficiency of the Construction. To obtain an estimate of the practical applicability of the complementation algorithm (Definition 3), we implemented it and applied it to randomly generated formulas. We generated a total of 10000 random formulae using the SPOT (Duret-Lutz et al. 2016) 2.11.3 tool `randltl` with 4 atomic propositions each. We then converted each of these formulas to Büchi automata using `ltl2tgba`. We used our prototypical tool to complement these automata with a timeout of 600 seconds and were successful in 99.47% of the cases. We then applied several optimizations to reduce the number of states in the complement, all of which maintained the good-for-MDP property. Table 1 provides statistics on our results, and Table 2 provides individual values for some example runs. The first 5 entries are the ones for which the complementation led to the largest number of states, while the next 5 were randomly selected

As seen in Table 1, the maximum number of complement states is more than a million, while the mean is much lower. The standard deviation is quite high. Looking at the data, this is because in most cases the number of states generated for the complement is relatively low, while in some cases it is very big. As seen, all optimizations lead to a reduction, although the effect of applying bisimulation lumping to all states in the end is not as large as the other ones. As seen in Table 2, in some cases the number of states was quite large. However, after applying the optimizations described, we were able to further reduce the number of states to make the resulting automaton suitable for model checking or RL.

Conclusion

Reinforcement learning often relies on the design of a suitable reward signal. While it's easy to design a reward signal as a function of the state and action for simpler problems, practical problems require non-Markovian rewards. We have introduced omega-regular decision processes (ODPs) as a formalism that provides great flexibility in specifying complex, non-Markovian rewards derived from a combination of qualitative and quantitative objectives. A key aspect of our approach is the ability for the decision maker to obtain rewards contingent upon the fulfillment of "promises" in the language of expressive ω-regular specifications. Our algorithm reduces the ODP optimization to a lexicographic optimization problem over MDPs with ω-regular and discounted objectives. This reduction is based on translating the collection semantics of promises to a good-for-MDPs Büchi automaton, which enables an automata-theoretic approach to optimization.

Acknowledgements

This work was supported in part by the EPSRC through grants EP/X017796/1 and EP/X03688X/1, the NSF through grant CCF-2009022 and the NSF CAREER award CCF-2146563; and the EU's Horizon 2020 research and innovation programme under grant agreement No 864075 (CAESAR).

References

Abadi, E.; and Brafman, R. I. 2021. Learning and Solving Regular Decision Processes. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*.

Amodei, D.; Olah, C.; Steinhardt, J.; Christiano, P.; Schulman, J.; and Mané, D. 2016. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*.

Babiak, T.; Křetínský, M.; Rehák, V.; and Strejcek, J. 2012. LTL to Büchi Automata Translation: Fast and More Deterministic. In *Tools and Algorithms for the Construction and Analysis of Systems*, 95–109.

Baier, C.; and Katoen, J.-P. 2008. *Principles of Model Checking*. MIT Press.

Bozkurt, A. K.; Wang, Y.; and Pajic, M. 2021. Model-Free Learning of Safe yet Effective Controllers. In *2021 60th IEEE Conference on Decision and Control (CDC), Austin, TX, USA, December 14-17, 2021*, 6560–6565. IEEE.

Brafman, R. I.; and De Giacomo, G. 2019. Planning for LTLf /LDLf Goals in Non-Markovian Fully Observable Nondeterministic Domains. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, 1602–1608.

Camacho, A.; Icarte, R. T.; Klassen, T. Q.; Valenzano, R. A.; and McIlraith, S. A. 2019. LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning. In *IJCAI*, volume 19, 6065–6073.

Camacho, A.; and McIlraith, S. A. 2019. Strong Fully Observable Non-Deterministic Planning with LTL and LTLf Goals. In *IJCAI*, 5523–5531.

Camacho, A.; Muise, C. J.; Baier, J. A.; and McIlraith, S. A. 2018. LTL Realizability via Safety and Reachability Games. In *IJCAI*, 4683–4691.

Clark, J.; and Amodei, D. 2016. Faulty Reward Functions in the Wild. https://openai.com/blog/faulty-reward-functions/. Accessed on: 01/18/2023.

Courcoubetis, C.; and Yannakakis, M. 1995. The Complexity of Probabilistic Verification. *J. ACM*, 42(4): 857–907.

Dimitrova, R.; Ghasemi, M.; and Topcu, U. 2018. Maximum realizability for linear temporal logic specifications. In *Automated Technology for Verification and Analysis: 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings 16*, 458–475. Springer.

Duret-Lutz, A.; Lewkowicz, A.; Fauchille, A.; Michaud, T.; Renault, E.; and Xu, L. 2016. Spot 2.0 - A Framework for LTL and ω-Automata Manipulation. In *Automated Technology for Verification and Analysis*, 122–129.

Filiot, E.; Jin, N.; and Raskin, J.-F. 2009. An antichain algorithm for LTL realizability. In *Computer Aided Verification: 21st International Conference, CAV 2009, Grenoble, France, June 26-July 2, 2009. Proceedings 21*, 263–277. Springer.

Finkbeiner, B.; and Schewe, S. 2013. Bounded synthesis. *International Journal on Software Tools for Technology Transfer*, 15(5-6): 519–539.

Friedgut, E.; Kupferman, O.; and Vardi, M. Y. 2006. Büchi Complementation Made Tighter. *Int. J. Found. Comput. Sci.*, 17(4): 851–868.

Fu, J.; and Topcu, U. 2014. Probably Approximately Correct MDP Learning and Control With Temporal Logic Constraints. In *Robotics: Science and Systems*.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. MIT Press.

Hahn, E. M.; Perez, M.; Schewe, S.; Somenzi, F.; Trivedi, A.; and Wojtczak, D. 2019. Omega-Regular Objectives in Model-Free Reinforcement Learning. In *Tools and Algorithms for the Construction and Analysis of Systems*, 395–412. LNCS 11427.

Hahn, E. M.; Perez, M.; Schewe, S.; Somenzi, F.; Trivedi, A.; and Wojtczak, D. 2020. Good-for-MDPs Automata for Probabilistic Analysis and Reinforcement Learning. In *Tools and Algorithms for the Construction and Analysis of Systems*, 306–323. LNCS 12078.

Hahn, E. M.; Perez, M.; Schewe, S.; Somenzi, F.; Trivedi, A.; and Wojtczak, D. 2023a. Mungojerrie: Linear-Time Objectives in Model-Free Reinforcement Learning. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 13993 of *Lecture Notes in Computer Science*, 527–545. Springer.

Hahn, E. M.; Perez, M.; Schewe, S.; Somenzi, F.; Trivedi, A.; and Wojtczak, D. 2023b. Omega-Regular Decision Processes. *CoRR*, abs/2312.08602.

Hahn, E. M.; Perez, M.; Schewe, S.; Somenzi, F.; Trivedi, A.; and Wojtczak, D. 2023c. Omega-Regular Reward Machines. In *ECAI 2023 - 26th European Conference on Artificial Intelligence, September 30 - October 4, 2023, Kraków, Poland - Including 12th Conference on Prestigious Applications of Intelligent Systems (PAIS 2023)*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, 972–979. IOS Press.

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

Icarte, R. T.; Klassen, T. Q.; Valenzano, R.; and McIlraith, S. A. 2022. Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, 73: 173–208.

Kupferman, O.; and Vardi, M. Y. 2001. Weak alternating automata are not that weak. *ACM Trans. Comput. Log.*, 2(3): 408–429.

McDaniel, M. A.; and Einstein, G. O. 2007. *Prospective memory: An overview and synthesis of an emerging field*. Sage Publications.

Mirhoseini, A.; Goldie, A.; Yazgan, M.; Jiang, J.; Songhori, E.; Wang, S.; Lee, Y.-J.; Johnson, E.; Pathak, O.; Bae, S.; et al. 2020. Chip placement with deep reinforcement learning. *arXiv preprint arXiv:2004.10746*.

Pan, A.; Bhatia, K.; and Steinhardt, J. 2022. The effects of reward misspecification: Mapping and mitigating misaligned models. *arXiv preprint arXiv:2201.03544*.

Pnueli, A. 1977. The Temporal Logic of Programs. In *IEEE Symposium on Foundations of Computer Science*, 46–57.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York, NY, USA: John Wiley & Sons.

Sadigh, D.; Kim, E.; Coogan, S.; Sastry, S. S.; and Seshia, S. A. 2014. A Learning Based Approach to Control Synthesis of Markov Decision Processes for Linear Temporal Logic Specifications. In *Conference on Decision and Control (CDC)*, 1091–1096.

Schewe, S. 2009a. Büchi Complementation Made Tight. In Albers, S.; and Marion, J., eds., *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, volume 3 of *LIPIcs*, 661–672. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany.

Schewe, S. 2009b. Tighter Bounds for the Determinisation of Büchi Automata. In de Alfaro, L., ed., *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5504 of *Lecture Notes in Computer Science*, 167–181. Springer.

Schewe, S.; Tang, Q.; and Zhanabekova, T. 2023. Deciding What Is Good-For-MDPs. In Pérez, G. A.; and Raskin, J., eds., *34th International Conference on Concurrency Theory, CONCUR 2023, September 18-23, 2023, Antwerp, Belgium*, volume 279 of *LIPIcs*, 35:1–35:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587): 484–489.

Skalse, J.; Howe, N. H.; Krasheninnikov, D.; and Krueger, D. 2022. Defining and characterizing reward hacking. *arXiv preprint arXiv:2209.13085*.

Somenzi, F.; and Bloem, R. 2000. Efficient Büchi Automata from LTL Formulae. In *Computer Aided Verification*, 248–263. LNCS 1855.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. MIT Press, second edition.

Wurman, P. R.; Barrett, S.; Kawamoto, K.; MacGlashan, J.; Subramanian, K.; Walsh, T. J.; Capobianco, R.; Devlic, A.; Eckert, F.; Fuchs, F.; et al. 2022. Outracing champion Gran Turismo drivers with deep reinforcement learning. *Nature*, 602(7896): 223–228.

Yuan, Y.; Yu, Z. L.; Gu, Z.; Deng, X.; and Li, Y. 2019. A novel multi-step reinforcement learning method for solving reward hacking. *Applied Intelligence*, 49: 2874–2888.