Omega-Regular Decision Processes

Ernst Moritz Hahn\textsuperscript{*1}, Mateo Perez\textsuperscript{*2}, Sven Schewe\textsuperscript{*3}, Fabio Somenzi\textsuperscript{2}, Ashutosh Trivedi\textsuperscript{2}, Dominik Wojtczak\textsuperscript{*3}

\textsuperscript{1}University of Twente, The Netherlands\textsuperscript{*}
\textsuperscript{2}University of Colorado Boulder, USA\textsuperscript{*}
\textsuperscript{3}University of Liverpool, UK\textsuperscript{*3}
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 \( \omega \)-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 \(-1\) (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 \( \omega \)-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, \( \omega \)-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.
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 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; 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 GF 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 MUNGOJERIE (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 com-plement-ation 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-

\begin{equation}
\mathcal{M}_o \text{ is a finite-state Markov chain. The behavior of } \mathcal{M} \text{ under a strategy } \sigma \text{ from } s \in S \text{ is defined on the probability space } (\text{Runs}_s^\mathcal{M}(s), \mathcal{F}_{\text{Runs}_s^\mathcal{M}(s)}, \Pr_\sigma(s)) \text{ over the set of in-


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

\lim_{N \to \infty} E_{\text{Disct}}(\lambda)^\mathcal{M}_s(s) \{ \sum_{1 \leq i \leq N} \lambda^{i-1} p(X_{i-1}, Y_i, X_i) \}. \quad (1)

\text{We define the optimal discounted reward } E_{\text{Disct}}^*\mathcal{M}_s(s) \text{ for a state } s \in S \text{ as } E_{\text{Disct}}^*\mathcal{M}_s(s) \equiv \sup_{\sigma \in \mathcal{L}\mathcal{M}} E_{\text{Disct}}(\lambda)^\mathcal{M}_s(s). \text{ A strategy } \sigma \text{ is discount-optimal if } E_{\text{Disct}}^*\mathcal{M}_s(s) = E_{\text{Disct}}^*\mathcal{M}_s(s) \text{ for all } s \in S. \text{ The optimal discounted cost can be computed in polynomial time (Puterman 1994).}

\text{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} = (Q, U, u_0, \delta, \rho) \text{ where } U \text{ is a finite set of states, } u_0 \in U \text{ is the starting state, } \delta: U \times U \rightarrow \mathbb{Q}^+ \text{ is the transition function, and } \rho: U \times U \rightarrow \mathbb{R} \text{ is the reward function. Given an MDP } \mathcal{M} = (S, s_0, A, T, \rho, L) \text{ and a reward \textbf{machine } } \mathcal{R} = (2^{AP}, U, u_0, \delta, \rho), \text{ their product } \mathcal{M} \times \mathcal{R} = (S \times U, (s_0, u_0), (A \times U), T', \rho^\mathcal{R}) \text{ is a rewardful MDP where the transition function } T'(s, u)(a, u')((s', u')) \text{ equals } T(s, a)(s') \text{ if } u' \in \delta(u, L(s)) \text{ equals 0 otherwise. Moreover, the reward function } \rho^\mathcal{R}(s, u)(a, u', s', u')) \text{ equals } \rho(u, L(s), u') \text{ if } (u, L(s), u') \in \delta \text{ and equals 0 otherwise. For discounted objectives, the optimal strategies of } \mathcal{M} \times \mathcal{R} \text{ are positional on } \mathcal{M} \times \mathcal{R}, \text{ inducing finite memory strategies over } \mathcal{M} \text{ maximizing the learning objective given by } \mathcal{R}.

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

\textbf{ω-Automata.} A (nondeterministic) Büchi automaton \textbf{(NBA)} is a tuple } \mathcal{A} = (\Sigma, Q, q_0, \delta, \gamma), \text{ where } \Sigma \text{ is a finite alphabet, } Q \text{ is a finite set of states, } \delta: \Sigma \times Q \rightarrow 2^Q \text{ is the transition function, and } \gamma: Q \times \Sigma \rightarrow 2^Q \text{ with } \gamma(q, \sigma) \subseteq \delta(q, \sigma)\)
for all \((q, \sigma) \in Q \times \Sigma\) are the accepting transitions. A run \(\rho\) of \(A\) on \(w \in \Sigma^*\) from the initial state \(q_0 \in Q\) is an \(\omega\)-word \(\rho_0, \rho_1, \rho_2, \ldots\) in \((Q \times \Sigma)^\omega\) such that \(\rho_0 = q_0\) and, for all \(i > 0\), \(\rho_i \in \delta(\rho_{i-1}, \sigma_{i-1})\). We write \(\inf(\rho)\) for the set of transitions that appear infinitely often in the run \(\rho\). A run \(\rho\) of an NBA \(A\) is accepting if \(\inf(\rho)\) contains a transition from \(\gamma\).

The language \(L(A, q)\) of \(A\) is the subset of words in \(\Sigma^*\) that have accepting runs in \(A\) from \(q\). A language is \(\omega\)-regular if it is accepted by a nondeterministic Büchi automaton.

A universal co-Büchi automaton (UCA) \(A = (\Sigma, Q, \delta, \gamma)\) is the dual of an NBA and its language can be defined using the notion of rejecting runs. We call a transition in \(\gamma\) rejecting and any runs with a transition in \(\gamma\) occurring infinitely often rejecting runs. The language \(L(A, q)\) of a UCA \(A\) is the set of \(\omega\)-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 \(M\) and a NBA automaton \(A\), the probabilistic model checking problem is to find a strategy that maximizes the probability of generating words in the language of \(A\). 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 \(A\) is called good-for-MDPs if, for any MDP \(M\), controlling \(M\) to maximize the chance that its trace is in the language of \(A\) and controlling the syntactic product \(\hat{M} \times A\) (defined next) to maximize the chance of satisfying the Büchi objective are the same. In other words, for any MDP, the nondeterminism of \(A\) can be resolved on-the-fly.

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

An end-component of an MDP \(M\) is a sub-MDP \(M'\) s.t. for every state pair \((s, s')\) in \(M'\) 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 Braffman 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 \(\omega\)-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 \(\omega\)-regular objectives and scalar rewards.

For an automaton of any type, an automaton schema \(A = (\Sigma, Q, \delta, F)\) (for DFA) or \(A = (\Sigma, Q, \delta, \gamma)\) (for NBAs or UCAs) is defined as an automaton without an initial state. For an automaton schema \(A = (\Sigma, Q, \delta, \gamma)\) and a state \(q \in Q\), we write \(A_q = (\Sigma, Q, q, \delta, \gamma)\) as the automaton with \(q\) as initial state and \(L(A, q)\) for its language. We express various transition guards using a DFA schema (lookback automaton) and various promises using a UCA schema¹ (lookahead automaton).

**Definition 1 (Omega-Regular Decision Processes).** An \(\omega\)-regular decision process (ODP) \(M\) is a tuple \((S, s_0, A, T, r, A_\text{a}, A_\text{b}, AP, L)\) 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,
- \(L : S \rightarrow 2^{AP}\) is the labeling function,
- \(A_\text{a} = (2^{AP}, Q, \delta, \gamma)\) is a lookback DFA schema,
- \(A_\text{b} = (2^{AP}, Q, \delta, \gamma)\) is a lookahead UCA schema,
- \(T : S \times Q \times A \times Q,a \rightarrow ])S\) is the transition function,
- and \(r : S \times Q \times A \times Q,a \rightarrow \mathbb{R}\) is the reward function.

An ODP with trivial lookahead \(L(A_\text{a}, q) = \Sigma^*\), for every \(q \in Q_\text{a}\), is a regular decision process (RDP). An ODP with trivial lookback \(L(A_\text{b}, q) = \Sigma^*\), for every \(q \in Q_\text{b}\), is a lookback 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 \(\langle s_0, (\beta_1, \sigma_1, \alpha_1), s_1, (\beta_2, \sigma_2, \alpha_2), \ldots \rangle \in S \times ((Q_\text{a} \times A \times Q_\text{a}) \times S)^\omega\) of \(M\) is an \(\omega\)-word such that \(\Pr(s_{i+1}|s_i, (\beta_{i+1}, \alpha_{i+1}, \alpha_{i+1}) > 0\) for all \(i \geq 0\).

A finite run is a finite such sequence. We say that a run \(\langle s_0, (\beta_1, \sigma_1, \alpha_1), s_1, (\beta_2, \sigma_2, \alpha_2), \ldots \rangle \in S \times ((Q_\text{b} \times A \times Q_\text{b}) \times S)^\omega\) is a valid run if for every \(i \geq 1\) we have that \(L(s_0) L(s_1) \ldots L(s_{i-1}) \in L(A_\text{b}, \beta_{i})\) and \(L(s_i)L(s_{i+1}) \ldots \in L(A_\text{a}, \alpha_i)\). The concepts of strategies, memory, and probability space are defined for the ODPs in an analogous manner to MDPs. We say that a strategy \(\sigma\) for an ODP is a valid strategy if the resulting runs are almost surely valid. Let \(\Pi_M\) be the set of all valid strategies of \(M\).

¹Why 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 McIrratha 2019) communities. They are often referred to as NBAs that recognize the complement language. It is worth noting that if an NBA \(A\) recognizes the models of an LTL or QPTL formula \(\phi\), or any other specification logic with negation, then \(\overline{A}\), 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 E\text{Disct}(\lambda)_{M}^{s}(s) for a strategy in an ODP \( M \) is defined as in (1). We define the optimal discounted reward E\text{Disct}_{\epsilon}^{s}(s) for a state \( s \in S \) as E\text{Disct}_{\epsilon}^{s}(s) = \sup_{\sigma \in \Pi_{M}^{s}} E\text{Disct}_{\epsilon}^{s}(s). A strategy \( \sigma \) is discount-optimal if E\text{Disct}_{\epsilon}^{s}(s) = E\text{Disct}_{\epsilon}^{s}(s) for all \( s \in S \). Given \( \epsilon > 0 \), we say that \( \sigma \) is \( \epsilon \)-optimal if E\text{Disct}_{\epsilon}^{s}(s) > E\text{Disct}_{\epsilon}^{s}(s) - \epsilon \) 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 lookahead is trivial \( L(A_0, q) = \Sigma^* \) and the lookahead is \( \Sigma^*(b\Sigma^*)^\omega \) (infinitely many \( b \)'s). While the achievable discounted reward of \( \lambda^t = \frac{1}{1-\epsilon} \) with any valid strategy, we can get arbitrarily close to this value by, e.g., choosing \( a \)'s until a reward \( > \frac{1}{1-\epsilon} - \epsilon \) is collected for any given \( \epsilon > 0 \), and henceforth choose \( b \)'s. While the optimal Büchi-discounted value is \( \frac{1}{1-\epsilon} \), no strategy can attain this value.

Throughout the rest of this paper, we will focus on the problem of computing optimal discounted values and \( \epsilon \)-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 \( M = (S, s_0, A, T, r, A_0) \), we can construct an MDP \( N = (S', s'_0, A', T', r') \) such that the optimal discounted value starting from \( s_0 \) in \( M \), denoted by E\text{Disct}_{\epsilon}^{s}(s_0), is equal to the optimal discounted value starting from \( s'_0 \) in \( N \), denoted by E\text{Disct}_{\epsilon}^{s}(s'_0). Moreover, a finite-memory optimal strategy for \( M \) can be computed from an optimal strategy for \( N \).

**Proof.** Simulating the lookahead automaton \( A_0 \) is a straightforward process. Without loss of generality, we can assume that (\( A_0, p \)) is deterministic for all \( p \in Q_0 \). We can simulate \( A_0 \) by computing, for each state \( p \in Q_0 \), the state \( \alpha(p) \in Q_0 \) that has been reached so far by \( (A_0, p) \) on the current prefix (if it exists; otherwise, \( \alpha(p) \) is undefined). A transition of \((S, s_0, A, T, r, A_0)\) with a lookahead \( r \in Q_0 \) can be triggered whenever \( F_0 \cap \alpha(r) \neq \emptyset \).

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 \( 2\text{EXPTIME} \)-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 \( 2\text{EXPTIME} \)-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 \( \text{EXPTIME} \)-hard for the former.

**Theorem 2** (Lower bounds). Finding an \( \epsilon \)-optimal strategy for a lookahead decision process \( M_{\text{cut}} = (S, s_0, A, T, r, A_0) \) is \( \text{EXPTIME} \)-hard in the size of \( A_0 \). If \( A_0 \) is given as an LTL formula, the problem becomes \( 2\text{EXPTIME} \)-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 \( \omega \)-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 \( \gamma \) 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 \( q_{0}^{f} \notin Q \) and \( Q' = Q \cup \{q_{0}^{f}\} \), we define the collection automaton \( \mathcal{C} = (\Sigma \times Q, Q', q_{0}^{f}, \delta', \gamma') \), 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

- \( \gamma'(q_{0}^{f}, (\sigma, q)) = \delta(q, \sigma) \) and \( \delta'(q_{0}^{f}, (\sigma, q)) = \{q_{0}^{f}\} \cup \delta(q, \sigma) \), that is, from the fresh initial state \( q_{0}^{f} \), we have a non-final transition back to \( q_{0}^{f} \) 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 \( q_0^\prime \) 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 \( \gamma'(q_0', (\sigma, S)) = \bigcup_{\sigma' \in S} \delta(q, \sigma') \) and \( \delta'(q_0', (\sigma, S)) = \gamma'(q, (\sigma, S)) \cup \delta(q_0') \); the proofs in this section are easy to adjust to this case.

**Theorem 3.** For a given UCA schema \( A \), the automaton \( C \) from above accepts a word \( w = (\sigma_0, q_0)(\sigma_1, q_1)(\sigma_2, q_2) \ldots \) 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 \( f : S \rightarrow \mathbb{N} \) from a finite set \( S \subseteq Q \) of states \( S \)-tight if, for some \( n \leq |S| \), it maps \( S \) to \( \{0, 1, \ldots , 2n-1\} \) and onto \( \{1, 3, \ldots , 2n-1\} \). We write \( T_S \) for the set of \( S \)-tight level-ranking functions. We call \( \text{rank}(f) = \max\{f(q) \mid q \in S\} \) (the \( 2n-1 \) from above) the rank of \( f \).

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

- \( Q' = Q_1 \cup Q_2 \) with \( Q_1 = 2^{\Sigma} \) and \( Q_2 = \{(S, O, f, i) \in 2^{Q} \times 2^{\Sigma} \times T_S \times \{0, 2, \ldots , 2n-2\} \mid O \subseteq f^{-1}(i)\} \)
- \( \delta' = \delta_1 \cup \delta_2 \cup \delta_3 \) with
  - \( \delta_1 : Q_1 \times \Sigma \rightarrow 2^{Q_1} \) with \( \delta_1(S, \sigma) = \{\delta(S, \sigma)\} \)
  - \( \delta_2 : Q_1 \times \Sigma \rightarrow 2^{Q_2} \) with \( (S', O, f, i) \in \delta_2(S, \sigma) \) iff \( S' = \delta(S, \sigma) , O = \emptyset \), and \( i = 0 \)
  - \( \delta_3 : Q_2 \times \Sigma \rightarrow 2^{Q_2} \) with \( (S', O', f', i') \in \delta_3((O, f, i), \sigma) \) iff the following holds:
    - \( S' = \delta(S, \sigma) \) and we define the auxiliary function \( g : S' \rightarrow 2^{0, \ldots , 2n-1} \) with \( g(q) \) equals
      - \( \{j \mid q \in \delta(f^{-1}(j), \sigma)\} \cup \{2 \cdot j / 2 \mid q \in \gamma(f^{-1}(j), \sigma)\} \)
    - \( f' \) 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 \( i' = (i+2) \mod (\text{rank}(f') + 1) \) and \( O' = f'^{-1}(i') \)
- \( \gamma' \) contains the transitions of \( \delta_3 \) from case (3) (the breakpoints) as well as transitions from \( \emptyset \).

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

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

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 \( C \) is good-for-MDPs, we take a different approach: we start from determinising the UCA \( A \) into a deterministic Streett automaton \( S \), using the standard determinisation from nondeterministic Büchi to deterministic Rabin automata (Schewe 2009b). It is then easy to see how an accepting run of \( S \) on a word can be simulated. The proof details are provided in the full version.

**Theorem 5.** For a given UCA \( A \), the NBA \( C \) 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 \( A \), the rank-based complementation algorithms from (Kupferman and Vardi 2001; Friedgut, Kupferman, and Vardi 2006; Schewe 2009a) provide good-for-MDP automata.

**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 \( \omega \)-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 \( \mathcal{M}_a = (\Sigma, s_0, A, T, r, \mathcal{A}_a) \) can be done in time polynomial in \( M \) and is \( \text{EXPTIME-complete} \) in the size of \( \mathcal{A}_a \) and \( 2\text{EXPTIME-complete} \) in the size of an LTL formula describing \( \mathcal{A}_a \).
with the lexicographic $\omega$ (Definition 3), we implemented it and applied it to randomly

Table 1: Statistics for randomly generated examples. $\text{orig}$: Number of states of the automaton generated by $\text{ltl2tgba}$, $\text{compl}$: Number of states of the complement, $\text{prune}$: Number of states after removing states with empty language, $\text{lumpd}$: Number of states after applying strong-bisimulation lumping in the final part of the automaton, $\text{lang}$: 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, $\text{lumpa}$: Number of states after applying strong bisimulation lumping for all states of the automaton, $\text{time}$: total time in seconds.

<table>
<thead>
<tr>
<th>formula</th>
<th>$\text{orig}$</th>
<th>$\text{compl}$</th>
<th>$\text{prune}$</th>
<th>$\text{lumpd}$</th>
<th>$\text{lang}$</th>
<th>$\text{lumpa}$</th>
<th>$\text{time}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Fd \lor (a \leftrightarrow Ga) &amp; (c \leftrightarrow Fb))$</td>
<td>14</td>
<td>25,107,909</td>
<td>16,585</td>
<td>2,120</td>
<td>115</td>
<td>60</td>
<td>269.61</td>
</tr>
<tr>
<td>${(c \xor Fd) &amp; F(b &amp; c)} &amp; Xd$</td>
<td>13</td>
<td>20,484,339</td>
<td>59,150</td>
<td>1,005</td>
<td>30</td>
<td>13</td>
<td>127.77</td>
</tr>
<tr>
<td>$F((a &amp; (1 \or (d \xor Xd))) &amp; (a &amp; c))$</td>
<td>10</td>
<td>19,317,020</td>
<td>18,540</td>
<td>103</td>
<td>40</td>
<td>29</td>
<td>167.43</td>
</tr>
<tr>
<td>$X(1 \or a) &amp; F(1 \or (c &amp; Xa))$</td>
<td>11</td>
<td>18,492,964</td>
<td>294,249</td>
<td>502</td>
<td>32</td>
<td>15</td>
<td>111.25</td>
</tr>
<tr>
<td>$G(Xa \xor (G(Gc &amp; Xd) &amp; Xd))$</td>
<td>14</td>
<td>18,129,540</td>
<td>9,152,588</td>
<td>909</td>
<td>80</td>
<td>73</td>
<td>112.71</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>formula</th>
<th>$\text{orig}$</th>
<th>$\text{compl}$</th>
<th>$\text{prune}$</th>
<th>$\text{lumpd}$</th>
<th>$\text{lang}$</th>
<th>$\text{lumpa}$</th>
<th>$\text{time}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$!(G(a &amp; c) \or Xd)$</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0.00</td>
</tr>
<tr>
<td>$XG(Gd \lor (a &amp; (c &amp; (M Ga))))$</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0.00</td>
</tr>
<tr>
<td>$!(b &amp; c) \lor (c &amp; Xa)$</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0.00</td>
</tr>
<tr>
<td>$(Ga \rightarrow b) \lor (a &amp; c)$</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>0.01</td>
</tr>
<tr>
<td>$(!c &amp; R Fb) \lor (Gd \rightarrow GFb)$</td>
<td>10</td>
<td>232,094</td>
<td>70,513</td>
<td>6,481</td>
<td>60</td>
<td>15</td>
<td>11.76</td>
</tr>
</tbody>
</table>

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 $\omega$-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 $\omega$-regular specifications. Our algorithm reduces the ODP optimization to a lexicographic optimization problem over MDPs with $\omega$-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


LTL and


