Mungojerrie: Linear-Time Objectives in Model-Free Reinforcement Learning*

Ernst Moritz Hahn[1], Mateo Perez[2], Sven Schewe[3], Fabio Somenzi[2](✉), Ashutosh Trivedi[2], and Dominik Wojtczak[3]

[1] University of Twente, Enschede, The Netherlands
[2] University of Colorado Boulder, Boulder, USA
fabio@colorado.edu
[3] University of Liverpool, Liverpool, UK

Abstract. Mungojerrie is an extensible tool that provides a framework to translate linear-time objectives into reward for reinforcement learning (RL). The tool provides convergent RL algorithms for stochastic games, reference implementations of existing reward translations for ω-regular objectives, and an internal probabilistic model checker for ω-regular objectives. This functionality is modular and operates on shared data structures, which enables fast development of new translation techniques. Mungojerrie supports finite models specified in PRISM and ω-automata specified in the HOA format, with an integrated command line interface to external linear temporal logic translators. Mungojerrie is distributed with a set of benchmarks for ω-regular objectives in RL.

1 Introduction

Reinforcement learning (RL) [41] is a sequential optimization approach where a decision maker learns to optimally resolve a sequence of choices based on feedback received from the environment. This feedback often takes the form of rewards and punishments proportional to the fitness of the decisions taken by the agent (or their effects) as judged by the environment towards some higher-level objectives. We call such objectives *learning objectives*. RL is inspired by the way dopamine-driven organisms latch on to past rewarding actions and hence, historically, RL adopted a myopic way of looking at the reward sequences in the form of the discounted-sum of rewards, where the discount factor controls the weight placed toward future rewards. More recently, other forms of reward aggregation, such as limit-average, have also been considered. A key design challenge for users of RL is that of translation: given a class of learning objectives and aggregator functions, design a reward function from the sequence of learner's choices to scalar rewards such that an RL agent maximizing the aggregated sum of rewards converges to an optimal policy for the learning objective.

* Mungojerrie is available at plv.colorado.edu/mungojerrie. This work is supported in part by the National Science Foundation (NSF) grant CCF-2009022 and by NSF CAREER award CCF-2146563. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements No 864075 (CAESAR) and 956123 (FOCETA).

© The Author(s) 2023
S. Sankaranarayanan and N. Sharygina (Eds.): TACAS 2023, LNCS 13993, pp. 527–545, 2023.
https://doi.org/10.1007/978-3-031-30823-9_27

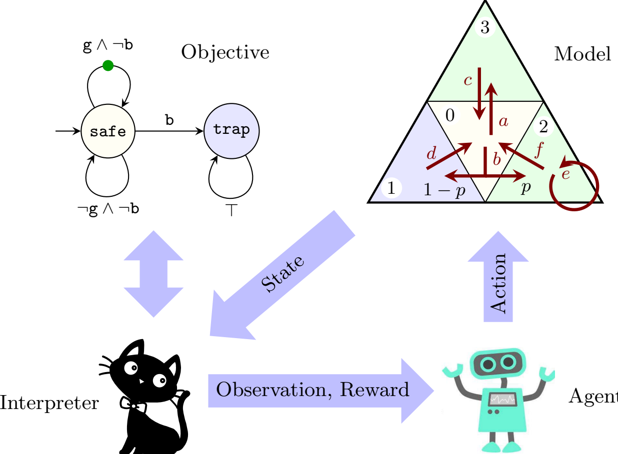

Fig. 1. The reinforcement learning loop implemented within Mungojerrie. The interpreter assigns reward to the agent based on the state of the model and automaton.

The translation of objectives to reward signals has historically been a largely manual process. Such translations not only depend on the expertise of the translator in reward engineering, they also pose obstacles to providing formal guarantees on the faithfulness of the translation. Unsurprisingly, specifying reward manually is prone to error [22,44]. As the practice of model-free RL continues to produce impressive results [38,31,29], the integration of RL in safety-critical system design is inevitable. An alternative to manually programming the reward function is to specify the objective in a formal language and have it "compiled" to a reward function. We call such a translation a *reward scheme*.

In designing reward schemes for RL, one strives to achieve an overall translation that is *faithful* (maximizing reward means maximizing the probability of achieving the objective) and *effective* (RL quickly converges to optimal strategies). While the faithfulness of a reward scheme can be established theoretically, its effectiveness requires experimental evaluation. Experimenting with reward schemes requires a framework for specifying learning objectives, environments, a wide range of RL algorithms, and an interface for connecting reward schemes with these components. In addition, it may be beneficial to have access to a probabilistic model checker to evaluate the quality of the policy computed by RL, and to compare it against ground truth.

> *Mungojerrie is designed to provide this functionality for learning requirements expressible as linear-time objectives (ω-regular languages [32] and linear temporal logic [27,33]) against finite MDPs and stochastic games.*

Features. Mungojerrie is designed with ease of use and extensibility in mind. Models in Mungojerrie can be specified in PRISM [25], which maintains compati-

bility with existing benchmarks, or by explicitly constructing the model via calls to internal functions. Mungojerrie supports reading ω-automata in the Hanoi Omega Automata (HOA) format [2], and has a command line interface connecting Mungojerrie with performant LTL translators (Spot [7] and Owl [24]). Mungojerrie provides an OpenAI Gym [4] like interface between the RL algorithms (included with the tool) and the learning environment to allow integration with off-the-shelf RL algorithms. The tool also has methods for performing probabilistic model checking (including end-component decomposition, stochastic shortest-path, and discounted-reward optimization) of ω-regular objectives on the same data structures used for learning. Mungojerrie also provides reference implementations of several reward schemes [11,12,14,19,23] proposed by the formal methods community. Mungojerrie is packaged with over 100 benchmarks and outputs GraphViz [8] for easy visualization of small models and automata.

An introductory example. Figure 2 shows an example MDP in which a gambler places bets with the aim of accumulating a wealth of 7 units. In addition the gambler will quit if her wealth wanes to just one unit more than once. This objective is captured by the (deterministic) Büchi automaton of Fig. 3. Mungojerrie computes a strategy for the gambler that maximizes the probability of satisfying her objective. Figure 4 shows the Markov chain that results from following this strategy. This figure was minimally modified from GraphViz output from Mungojerrie. Note that the strategy altogether avoids the state in which $x = 1$; hence it achieves the same probability of success (5/7) as an optimal strategy for the simpler objective of eventually reaching $x = 7$ (without going broke). Mungojerrie computes the strategy of Fig. 4 by RL; it can also verify it by probabilistic model checking.

2 Overview of Mungojerrie

Models. The systems used in Mungojerrie consist of finite sets of states and actions, where states are labeled with atomic propositions. There are at most two strategic players: Max player and Min player. Each state is controlled by one player. We call models where all states are controlled by Max player Markov decision processes (MDPs) [34]. Else, we refer to them as stochastic games [5].

Mungojerrie supports parsing models specified in the PRISM language. The allowed model types are “mdp” (Markov decision process) and “smg” (stochastic multiplayer game) with two players. There should be one initial state. The interface for building the model is exposed, allowing extensions of Mungojerrie to connect with parsers for other languages. The authors of [6] used Mungojerrie in their experiments by extending the tool to support continuous-time MDPs.

Properties. The properties natively supported by Mungojerrie are ω-regular languages. Starting from the initial state, the players produce an infinite sequence of states with a corresponding infinite sequence of atomic propositions: an ω-word. The inclusion of this ω-word in our ω-regular language determines

whether or not this particular run satisfies the property. The Max player maximizes the probability that a run is satisfying, while goal of the Min player is the opposite.

We specify our ω-regular language as an ω-automaton, which may be nondeterministic. For model checking and RL, this nondeterminism must be resolved on the fly. Automata where this can be done in any MDP without changing acceptance are said to be Good-for-MDPs (GFM) [13]. Automata where this can be done in any stochastic game without changing acceptance are said to be Good-for-Games (GFG) [21]. In general, nondeterministic Büchi automata are not GFM, but two classes of GFM Büchi automata with limited nondeterminism have been studied: suitable limit-deterministic Büchi automata [10,37] and slim Büchi automata [13].

The user of Mungojerrie can either provide the ω-automaton directly or use one of the supported external translators to generate the automaton from LTL with a single call to Mungojerrie. Mungojerrie reads automata specified in the HOA format. Mungojerrie supports providing the ω-automaton directly for testing the effectiveness of different automata for learning (see Section 4). The LTL translators that can be called from Mungojerrie are the ePMC plugin from [13], Spot [7], and Owl [24] for generating slim Büchi, deterministic parity, and suitable limit-deterministic Büchi automata. The user is responsible for the ω-automata provided directly having the appropriate property, GFM or GFG.

For use in Mungojerrie, the labels and acceptance conditions for the automaton should be on the transitions. The acceptance conditions supported by

```
0  mdp
1
2  const int Wealth = 5;    // initial gambler's wealth
3  const double p   = 1/2; // probability of winning one bet
4
5  label "rich" = x = 7;
6  label "poor" = x = 1;
7
8  module gambler
9    x : [0..7] init Wealth;
10
11   [b0] x=0 ∨ x=7 → true; // absorbing states
12   [b1] x>0 ∧ x<7 → p : (x'=x+1) + (1−p) : (x'=x−1);
13   [b2] x>1 ∧ x<6 → p : (x'=x+2) + (1−p) : (x'=x−2);
14   [b3] x>2 ∧ x<5 → p : (x'=x+3) + (1−p) : (x'=x−3);
15 endmodule
```

Fig. 2. A Gambler's Ruin model in the PRISM language. Line 13, for example, says that when $1 < x < 6$, the gambler may bet two units because action b2 is enabled. The '+' sign does double duty: as addition symbol in arithmetic expressions and as separator of probabilistic transitions.

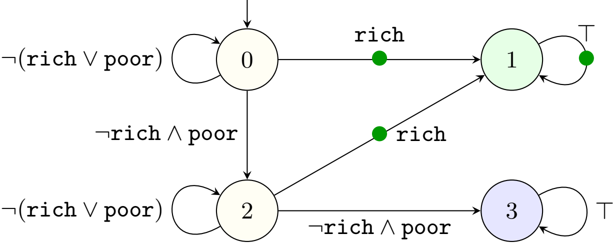

Fig. 3. Deterministic Büchi automaton equivalent to the LTL formula $\neg\texttt{poor}\,\mathsf{U}\big(\texttt{rich} \vee (\texttt{poor} \wedge \mathsf{X}(\neg\texttt{poor}\,\mathsf{U}\,\texttt{rich}))\big)$. The transitions marked with the green dots are accepting.

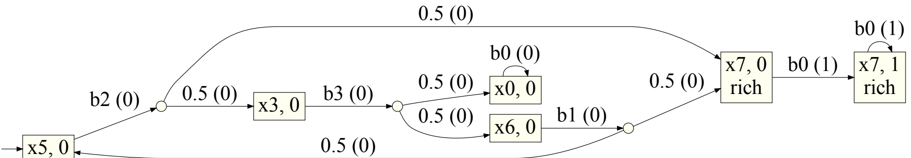

Fig. 4. Optimal gambler strategy for the objective of Fig. 3. Boxes are decision states and circles are probabilistic choice states. For a decision state, the label gives the value of x and the state of the automaton. Transitions are labelled with either an action or a probability, and with the priority (1 for accepting and 0 for non-accepting).

Mungojerrie should be reducible to parity acceptance conditions without altering the transition structure of the automaton. This includes parity, Büchi, co-Büchi, Streett 1 (one pair), and Rabin 1 (one pair) conditions. Nondeterministic automata must have Büchi acceptance conditions. Generalized acceptance conditions are not supported in version 1.1.

Reinforcement Learning. The RL algorithms optimize over MDP/Stochastic game environments equipped with a Markovian reward function. The reward function assigns a reward $R_{t+1} \in \mathbb{R}$ dependent on the state and action at timestep t and the next state at timestep $t+1$. As the players make their choices within the environment, the resulting play produces a sequence of states, actions, and rewards $(S_0, A_0, R_1, S_1, A_1, R_2, \ldots)$. The discounted reward aggregator is

$$\mathrm{disc}_\gamma(\pi, \nu) = \mathbb{E}_{\pi,\nu}\Big[\sum_{t \geq 0} \gamma^t R_{t+1}\Big] \enspace ,$$

where π is the strategy for Max player, ν is the strategy for Min player, $\gamma \in [0, 1)$ is the discount factor, and R_t is the reward at timestep t. We can set $\gamma = 1$ when

with probability 1 we enter an absorbing sink (termination), where we receive no reward. This is called the episodic setting. Another well-studied RL aggregator is the limit-average reward defined as

$$\text{avg}(\pi, \nu) = \limsup_{n \to \infty} \frac{1}{n} \mathbb{E}_{\pi,\nu} \Big[\sum_{n \geq t \geq 0} R_{t+1} \Big].$$

The limit-average reward aggregator is natural in the continuing setting, where the agent's trajectory is never reset and there is no preferred initial state [30]. The objective of RL is to compute the optimal value and policies for a given aggregator. Mungojerrie includes the stochastic game extensions of Q-learning [43], Double Q-learning [20], and Sarsa(λ) [40] for RL in finite state and action models. Mungojerrie also includes Differential Q-learning [42] for average RL in finite communicating MDPs. We collectively refer to parameters that are set by hand prior to running an RL algorithm as hyperparameters. Mungojerrie supports changing all hyperparameters from the command line. As the design of Mungojerrie separates the learning agent(s) from the reward scheme, extending Mungojerrie to include another RL algorithm is easy.

Reward Schemes. The user of Mungojerrie can either select one of the reward schemes included with the tool or extend the tool to include a new reward scheme. Mungojerrie also allows the use of the reward specified in the PRISM model (either state- or action-based). The following reward schemes are included in version 1.1 of Mungojerrie:

– Limit-reachability. The *limit-reachability* scheme [11] uses a GFM Büchi automaton. This reward scheme converts accepting edges in the automaton into a transition to a sink with probability $1-\zeta$ with a reward of $+1$, where $0 < \zeta < 1$ is a hyperparameter. All other transitions produce zero reward. For a sufficiently large ζ and discount factor γ, strategies that are optimal for the discounted reward maximize the probability of satisfaction of the Büchi objective.
– Multi-discounted. The multi-discounted reward scheme [3] also uses a GFM Büchi automaton. This translation converts accepting edges in the automaton into a transition that gives $1-\gamma_B$ reward with a discount of γ_B, where $0 < \gamma_B < 1$ is a hyperparameter. All other transitions yield no reward and are discounted by the standard discount factor γ. For suitably large γ_B and γ, discounted reward optimal strategies maximize the probability of satisfaction of the Büchi objective.
– Dense limit-reachability. The dense limit-reachability reward scheme [12] connects the approaches of [11] and [3]. This reward scheme is identical to [11] except for giving a $+1$ reward given every time an accepting transition is seen, instead of only when the transition to the sink succeeds. Since discounting can be thought of as a constant stopping probability [41], this reward scheme is the same in expectation as a scaled version of [3].
– Parity. The parity reward scheme was proposed for stochastic games in [14]. For two-player games, it requires a GFG automaton. This translation utilizes a deterministic parity automaton with a max odd objective. Transitions of priority

i go to a sink with probability ε^{k-i}, where k is the number of priorities and $0 < \varepsilon < 1$ is a hyperparameter. The transition to the sink receives a $+1$ or -1 reward for odd or even priorities, respectively. All other transitions receive a zero reward. For sufficiently small ε, maximizing the cumulative reward results in a strategy maximizing the probability of satisfaction of the parity objective.

– Priority tracker. The priority tracker reward scheme was proposed by Hahn et al. [14]. For MDPs, Hahn et al. introduce a priority tracker gadget that takes a parity objective with a hyperparameter $0 < \varepsilon < 1$. The priority tracker consists of two stages. In stage one, we wait for transients to end by ending the stage with probability ε on each step. In the second stage, we detect the maximum priority occuring infinitely often with a set of wait states, where we accept the current maximum with probability ε on each step. For sufficiently small ε and large discount γ, maximizing the discounted reward also maximizes the probability of satisfaction of the parity objective.

– Lexicographic. Hahn et al.[19] proposed this reward scheme for lexicographic ω-regular objectives. In this reward scheme, there is a tracker gadget that keeps track of which accepting edges for the GFM Büchi automata have been seen. When the tracker indicates that at least one accepting edge has been seen, the learning agent can decide to "cash in" the tracker, which clears the tracker. When this happens, with probability $1 - \zeta$ the learning agent receives a reward which is the weighted sum of seen accepting edges, scaled by powers of f, and transitions to a terminating sink, where $0 < \zeta < 1$ and $f \geq 1$ are hyperparameters. For suitable f, ζ, and γ, maximizing the discounted reward yields the lexicographically optimal strategy.

– Average. The average reward scheme [23] translates absolute liveness ω-regular objectives, which means the objective is concerned with eventual satifaction, to average reward for communicating MDPs. Given a GFM Büchi automaton, transitions from every state in the automaton back to the initial state are introduced, so called "resets". A hyperparameter $c < 0$ is introduced which gives a penalizing reward to these resets. Accepting edges are then given a reward of $+1$. Positional policies that maximize the average reward also maximize the probability of satisfaction of the objective.

– Reward on accept. This reward scheme was proposed in [35]. The translation of [35] picks a pair in a Rabin automaton to satisfy, and gives positive and negative reward for the good and bad states of the pair, respectively. In general, picking the winning pair ahead of time is not possible [11]. For a Büchi automaton, this corresponds to giving positive ($+1$) rewards for accepting edges and zero rewards otherwise. While this reward scheme was shown to be not faithful [11] for general objectives, it is included for comparison purposes.

3 Tool Design

The primary design goal of Mungojerrie is to enable extensibility. To accomplish this, Mungojerrie separates different processing stages as much as possible so that extensions can reuse other components. We begin by presenting the architecture

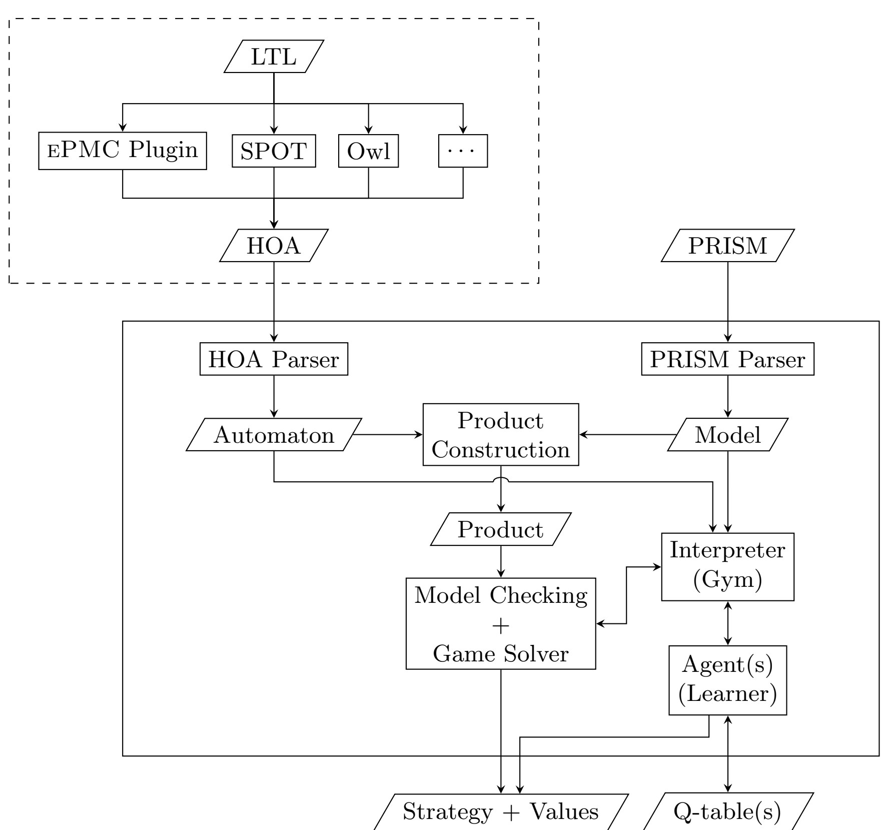

Fig. 5. Architecture of Mungojerrie 1.1.

of Mungojerrie. Afterwards, we take a closer at the novel slim Büchi automata plugin, which is described here in detail for the first time.

Architecture of Mungojerrie. Mungojerrie begins its execution by parsing the input PRISM and HOA (see upper part of Fig. 5). The HOA is either read in from a file or piped from a call to one of the supported LTL translators. In particular the EPMC plugin from [13], an LTL translator capable of producing slim Büchi automata, is packaged with the tool. Requested automaton modifications, such as determinization, are run after this step. If specified, Mungojerrie creates the synchronous product between the automaton and the model, and runs model checking or game solving [1,15,16]. The requested strategy and values are returned. Due to this step, Mungojerrie has been connected to external linear program solvers. This enabled the extension of Mungojerrie to compute reward maximizing policies via a linear program for branching Markov decision processes in [18].

If learning has been specified, the interpreter takes the automaton and model, without explicitly forming the product, and provides an interface akin to OpenAI Gym [4] for the RL agent to interact with the environment and receive rewards. When learning is complete, the Q-table(s) can be saved to a file for later use, and the interpreter forms the Markov chain induced by the learned strategy and passes it to the internal model checker for verification.

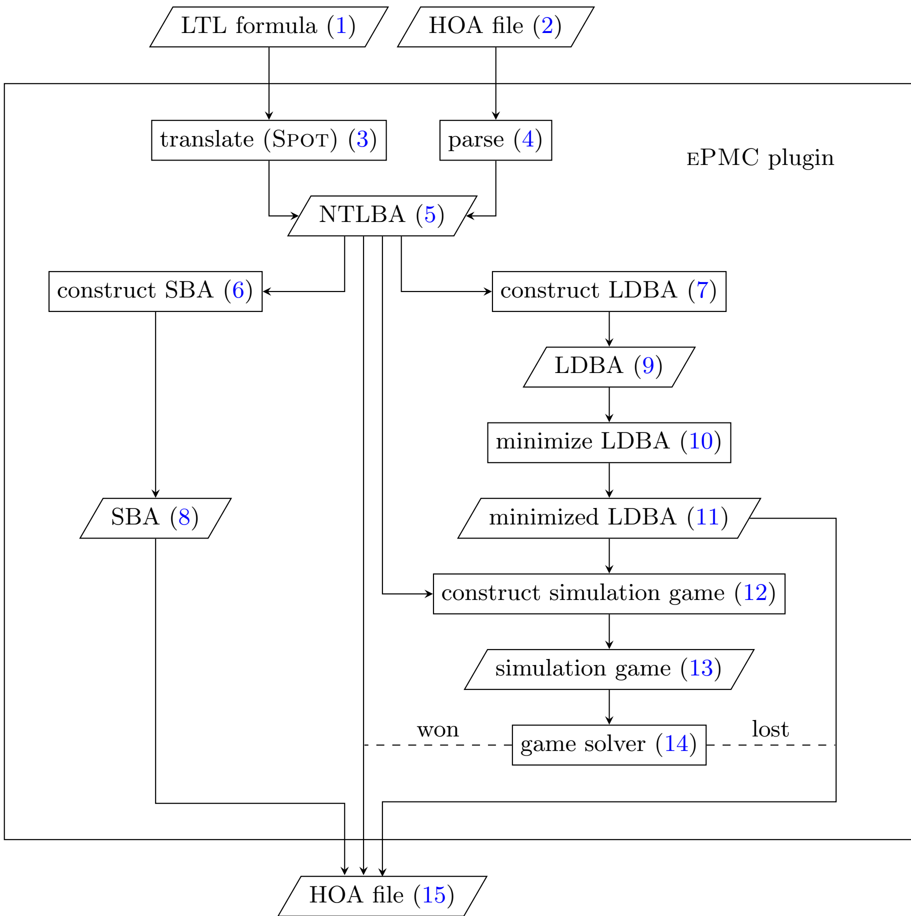

Fig. 6. Automata generation block diagram

Slim Büchi Automata Generation. For reward schemes involving LTL, the ω-regular automata translation is an important part of the design. Certain automata may be more effective for learning than others. Slim Büchi automata [13] were designed with learning considerations in mind. The translator that

produces these automata is packaged with Mungojerrie. We will now describe its design in detail for the first time.

We have implemented slim Büchi automata generation as a plugin of the probabilistic model checker ePMC [17]. The process is described in Fig. 6. The starting point is a transition-labeled Büchi automaton in HOA format [2] (2) or an LTL formula (1). In case we are given an automaton in HOA format, we parse this automaton (4) and if we are given an LTL formula, we use the tool Spot [7] to transform the formula into an automaton (3). In both cases, we end up with a transition-labeled Büchi automaton (5).

Afterwards, we have two options. The first option is to transform (6) this automaton into a slim Büchi automaton (8) [13]. These automata can then be directly composed with MDPs for model checking or used to produce rewards for learning. The other option is to construct (7) a suitable limit-deterministic Büchi automaton (SLDBA) (9). Automata of this type consist of an initial part and a final part. A nondeterministic choice only occurs when moving from the initial to the final part by an ε transition (a transition without reading a character). SLDBA can be directly composed with MDPs. However, SLDBA directly constructed from general Büchi automata are often quite large, which in turn also means that the product with MDPs would be quite large as well. Therefore, we have implemented further optimization steps. We can apply a number of algorithms to minimize (10) this automaton so as to achieve a smaller SLDBA (11). To do so, we implemented several methods:

- Subsuming the states in the final part with an empty language
- Signature-based strong bisimulation minimization in the final part
- Signature-based strong bisimulation minimization in the initial part
- Language-equivalence of states in the final part
- If we have a state s in the initial part for which we find a state s' in the final part where the language of s and s' are the same, we can remove all transitions of s and add an ε transition from s to s' instead. Afterwards, automaton states that cannot be reached anymore can be removed.

Each of these methods has a different potential for minimization as well as runtime. We therefore allow to specify which optimizations are to be used and in which order they are applied.

Once we have optimized the SLDBA, we could directly use it for later composition with an MDP. Another possibility is to prove that the original automaton is already good for MDPs. If this is the case, then it is often preferable to use the original automaton: being constructed by specialized tools such as Spot, it is often smaller than the minimized SLDBA. The original automaton is good-for-MDPs if it *simulates* the SLDBA [13]. If it does, then it is also composable with MDPs. Otherwise, it is unknown whether it is suitable for MDPs. In this case, sometimes more complex notions of simulation can be used, but existing decision procedures are too expensive to implement [36].

To show simulation, we construct (12) a simulation game, which in our case is a transition-labeled parity game (13) with 3 colors. We solve these games using (a slight variation of) the McNaughton algorithm [28]. (We are aware

that specialized algorithms for parity games with 3 colors exist [9]. However, so far the construction of the arena, not solving the game, turned out to be the bottleneck here). If the even player is winning, the simulation holds. Otherwise, more complex notions of simulation can be used, which however lead to larger parity games being constructed. In case the even player is winning for any of them, we can use the original automaton, otherwise we have to use the SLDBA. In any case, we export the result to an HOA file (15). For illustration and debugging , automata and simulation games can be exported to the GraphViz [8].

4 Case Studies

To showcase how Mungojerrie can be used to experiment with different reward schemes, we provide three case studies. In the first case study, we demonstrate how Mungojerrie can be used to compare the effectiveness of two different reward schemes on the same system. In the second case study, we consider the design space of automata, and demonstrate how Mungojerrie can be used to compare how different ω-automata change learning effectiveness. This is important for considering how to design LTL translators that produce automata that are effective for learning. In the last case study, we demonstrate how the different outputs of Mungojerrie can be used. For additional experimental results obtained using Mungojerrie, we refer readers to [11,12,14,19,39,45,23] for case studies testing ω-regular reward schemes, and [13] for the EPMC plugin. We also refer readers to [26, Fig. 3] which examined RL for scLTL properties, [6] for continuous-time MDPs, and [18], which extended Mungojerrie to test model-free reinforcement learning in branching Markov decision processes.

4.1 Comparing Reward Schemes

To demonstrate how Mungojerrie may be used to compare reward schemes, we compare the reward scheme of [11] with a modification of it that assigns a +1 reward on every accepting edge, as introduced in [12]. We compare these two methods on the same problem, where the learner must safely navigate two robots on a slippery gridworld to a goal. We also fix the problem parameters $\zeta = 0.99$ and $\gamma = 0.99999$, and the use of Q-learning. Since we are interested in which method will converge sooner, we fix the amount of training to be relatively low. We allow the two parameters specific to Q-learning, the learning rate α and the exploration rate ε, to be varied in order to find the optimal combination for each method. We average 10 runs for each grid point. This required 32000 runs, which took approximately 79 CPU hours (single-core) on a 2.5GHz Intel Xeon E5-2680 v3. This corresponds to an average of approximately 188000 sampled transitions per second per core, including model checking time. This sampling rate is typical of what was observed in other experiments.

Figure 7 shows the probability of satisfaction of the learned strategy as computed by the model checker of Mungojerrie. One can see that under these conditions, the reward scheme from [12] is able to consistently learn probability

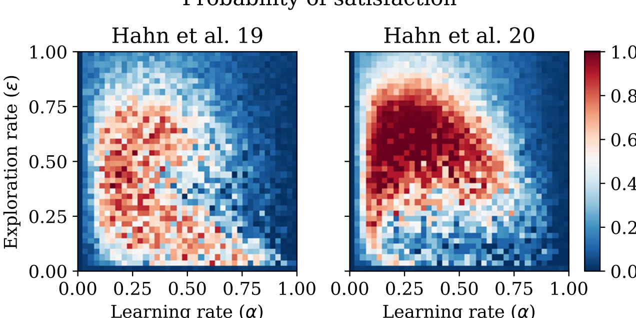

Fig. 7. Probability of satisfaction of learned strategies as computed by the model checker of Mungojerrie. 'Hahn et al. 19' refers to the translation of [11]. 'Hahn et al. 20' refers to the translation of [12] that assigns +1 reward on every accepting edge with reachability parameter ζ. Each grid point is the average of 10 runs.

1 strategies under certain parameter combinations, while [11] does not. Figure 8 shows the difference in the estimated probability of satisfaction, found by taking the value from the initial state of Q-table and renormalizing it appropriately, and the probability of satisfaction of the learned strategy computed by the model checker of Mungojerrie. One can see that the reward scheme of [11] sometimes overestimates and sometimes underestimates when it achieves a high actual probability of satisfaction under these conditions. However, on the same example, the reward scheme of [12] consistently underestimates everywhere. In summary, Mungojerrie allowed us to see that, although the reachability reward scheme of [12] may achieve higher probabilities of satisfaction sooner, it may take longer for the values in the Q-table to properly converge.

4.2 Comparing Automata

An ω-regular objective may be described by different automata, many of which may be good-for-MDPs. Mungojerrie can be used to compare the effectiveness of such automata when used in RL. Consider the two nondeterministic Büchi automata shown in Fig. 9. Both are equivalent to the LTL formula $(\mathsf{F}\,\mathsf{G}\,x) \vee (\mathsf{G}\,\mathsf{F}\,y)$, but the one on the right should be better for learning: long transient sequences of observations that satisfy $x \wedge \neg y$ may convince the agent to commit to State 1 of the left automaton too soon.

To test this conjecture, we specified a model in PRISM organized in two long chains. In one of them the agent sees many xs for a while, but eventually only sees ys. In the other chain the situation is reversed. Which chain is followed is up to chance. We then used the reward scheme from [3] with Q-learning under the default hyperparameters in Mungojerrie, $\gamma_B = 0.99$, $\gamma = 0.99999$, $\alpha = 0.1$, and $\varepsilon = 0.1$. We then trained for 20000 episodes under each automaton, and used Mungojerrie to compute the probability of satisfaction of the property at periodic

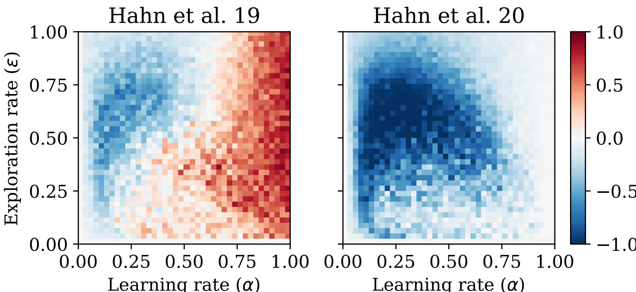

Fig. 8. Estimated probability of satisfaction of learned strategies minus the probability of satisfaction computed by the model checker of Mungojerrie. Blue indicates underestimation, while red indicates overestimation. Hahn et al. 19 refers to the translation of [11]. Hahn et al. 20 refers to the translation of [12] that assigns +1 reward on every accepting edge with reachability parameter ζ. Each grid point is the average of 10 runs.

intervals. Since learning to control the left automaton requires thorough and deep exploration, we conjectured that optimistic intialization of the Q-table [41] to the value 0.8 will improve performance. We took the average of 1000 runs for each combination.

Figure 10 shows the resulting curve. When using the LDBA without optimistic intialization, the learning agent is unable to learn the optimal strategy under these conditions. While it is worth noting that using the LDBA without optimistic initialization eventually converges to the optimal strategy with enough training, it is clear that the choice of the automaton can have a significant impact on learning performance. Therefore, the design of translations from LTL to automata has a role to play in producing effective reward schemes.

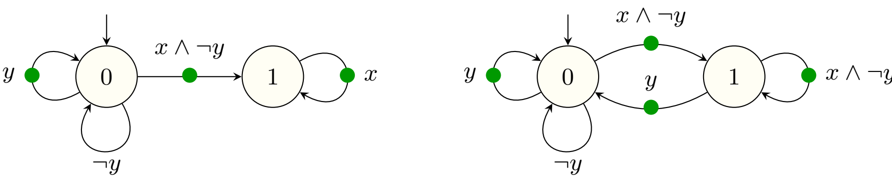

Fig. 9. Equivalent, but not equally effective, Büchi automata. “LDBA” and “Forgiving” refer to the automaton the left and right, respectively.

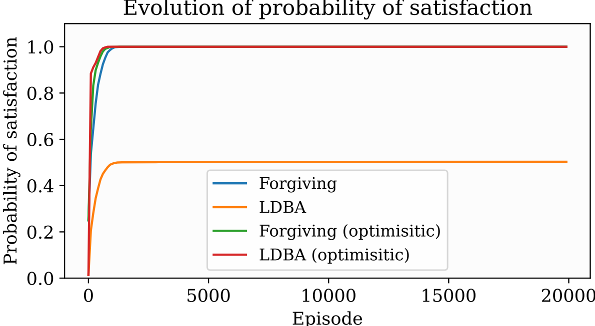

Fig. 10. Plot of the evolution of the probability of satisfaction of learned strategies as computed by the model checker of Mungojerrie. "Forgiving" and "LDBA" refer to the left and right automata in Figure 9, respectively. "(optimistic)" indicates optimistic initialization of the Q-table was used. Each curve is the average of 1000 runs.

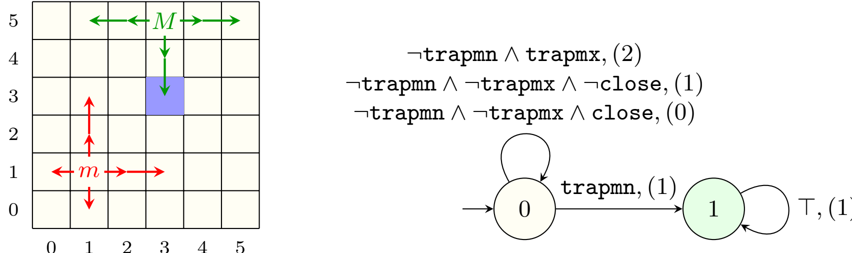

Fig. 11. A grid-world stochastic game arena (left) and a deterministic parity automaton for the objective (right).

4.3 A Game of Pursuit

Figure 11 describes a stochastic parity game of pursuit in which the Max player (M) tries to escape from the Min player (m). At each round, each player in turn chooses a direction to move. If movement in that direction is not obstructed by a wall, then the player moves either two squares or one square with equal probabilities. One square of the grid is a trap, which m must avoid at all times, but M may visit finitely many times. Player M should be at least 5 squares away from player m infinitely often. This objective is described by the LTL property $(\mathsf{F}\,\neg\mathtt{trapmn}) \vee ((\mathsf{F}\,\mathsf{G}\,\neg\mathtt{trapmx}) \wedge (\mathsf{G}\,\mathsf{F}\,\neg\mathtt{close}))$, where `trapmn` and `trapmx` are true when m and M visit the trap square, respectively, and `close` is true when the Manhattan distance between the two players is less than 5 squares. This objective translates to the deterministic parity automaton in Fig. 11, which accepts a word if the maximum recurring priority of its run is odd.

Unlike the example of Fig. 2, inspection of the Markov chain induced by an optimal strategy and manual verification of the optimality of the learned

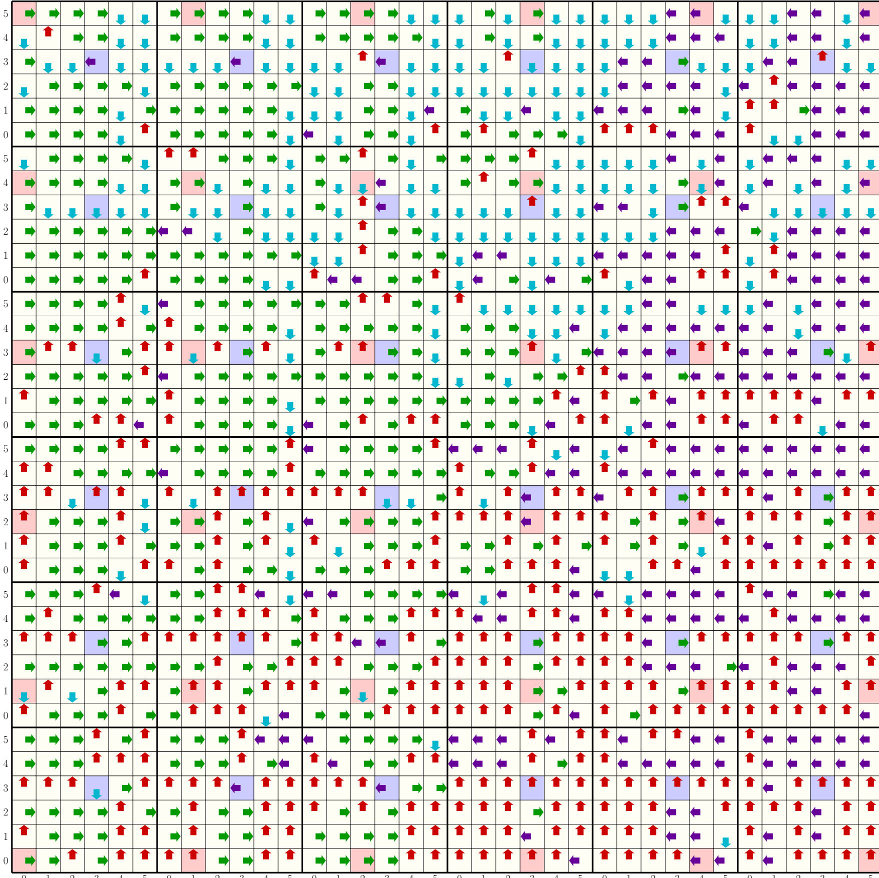

Fig. 12. Max player learned strategy for the game of Fig. 11 when the automaton is in State 0. (Any strategy will do when the automaton is in State 1.) In each 6×6 box the rose-colored square is the position of the minimizing player, while the light-blue square marks the trap.

strategy is impractical. Instead, the model checker of Mungojerrie has verified the optimality of this strategy from the intial state. For visualization, Mungojerrie can also save the strategy in CSV format. Postprocessing can then produce a graphical representation like the one of Fig. 12. The color gradient shows that, in the main, M's strategy is to move away from m.

5 Conclusion

We have introduced Mungojerrie, an extensible tool for experimenting with reward schemes for RL, with a focus on ω-regular objectives. Mungojerrie allows the specification of models in PRISM [25] and ω-automata in HOA [2]. Mul-

tiple LTL translators can be called from the tool [7,24], including the ePMC plugin introduced in [13] for the construction of slim Büchi automata. Mungojerrie includes various reward schemes [11,3,12,14,19,23,35] for ω-regular objectives and model-free RL algorithms [43,20,40,23]. Mungojerrie also includes an internal probabilistic model checker for the verification of learned strategies against ω-regular objectives, and for allowing users to verify that developed examples are as intended. The tool also comes packaged with benchmarks for ω-regular objectives in RL.

We have discussed Mungojerrie's design and demonstrated how Mungojerrie can be used to perform comparisons of reward schemes for ω-regular objectives. The source and documentation of Mungojerrie are publicly available.

References

1. de Alfaro, L.: Formal Verification of Probabilistic Systems. Ph.D. thesis, Stanford University (1998)
2. Babiak, T., Blahoudek, F., Duret-Lutz, A., Klein, J., Křetínský, J., Müller, D., Parker, D., Strejček, J.: The Hanoi omega-automata format. In: Computer Aided Verification (CAV). pp. 479–486 (2015), LNCS 9206
3. Bozkurt, A.K., Wang, Y., Zavlanos, M.M., Pajic, M.: Control synthesis from linear temporal logic specifications using model-free reinforcement learning. In: 2020 IEEE International Conference on Robotics and Automation (ICRA). pp. 10349–10355 (2020). https://doi.org/10.1109/ICRA40945.2020.9196796
4. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI Gym. CoRR **abs/1606.01540** (2016)
5. Condon, A.: The complexity of stochastic games. Inf. Comput. **96**(2), 203–224 (1992)
6. Dole, K., Gupta, A., Komp, J., Krishna, S.N., Trivedi, A.: Event-triggered and time-triggered duration calculus for model-free reinforcement learning. In: 42nd IEEE Real-Time Systems Symposium, RTSS 2021, Dortmund, Germany, December 7-10, 2021. pp. 240–252. IEEE (2021). https://doi.org/10.1109/RTSS52674.2021.00031,
7. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., Xu, L.: Spot 2.0 — a framework for LTL and ω-automata manipulation. In: Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16). Lecture Notes in Computer Science, vol. 9938, pp. 122–129. Springer (Oct 2016)
8. Ellson, J., Gansner, E.R., Koutsofios, E., North, S.C., Woodhull, G.: Graphviz and dynagraph - static and dynamic graph drawing tools. In: Jünger, M., Mutzel, P. (eds.) Graph Drawing Software, pp. 127–148. Springer (2004)
9. Etessami, K., Wilke, T., Schuller, A.: Fair simulation relations, parity games, and state space reduction for Büchi automata. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) Automata, Languages and Programming: 28th International Colloquium. pp. 694–707. Springer, Crete, Greece (Jul 2001), lNCS 2076
10. Hahn, E.M., Li, G., Schewe, S., Turrini, A., Zhang, L.: Lazy probabilistic model checking without determinisation. In: Concurrency Theory, (CONCUR). pp. 354–367 (2015)

11. Hahn, E.M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., Wojtczak, D.: Omega-regular objectives in model-free reinforcement learning. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 395–412 (2019), LNCS 11427
12. Hahn, E.M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., Wojtczak, D.: Faithful and effective reward schemes for model-free reinforcement learning of omega-regular objectives. In: ATVA: Automated Technology for Verification and Analysis. pp. 108–124 (2020), LNCS 12302
13. Hahn, E.M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., Wojtczak, D.: Good-for-MDPs automata for probabilistic analysis and reinforcement learning. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 306–323 (2020), LNCS 12078
14. Hahn, E.M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., Wojtczak, D.: Model-free reinforcement learning for stochastic parity games. In: CONCUR: International Conference on Concurrency Theory. pp. 21:1–21:16 (Sep 2020), LIPIcs 171
15. Hahn, E.M., Schewe, S., Turrini, A., Zhang, L.: A simple algorithm for solving qualitative probabilistic parity games. In: Computer Aided Verification. pp. 291–311. Part II (2016), LNCS 9780
16. Hahn, E.M., Schewe, S., Turrini, A., Zhang, L.: Synthesising strategy improvement and recursive algorithms for solving 2.5 player parity games. In: Verification, Model Checking, and Abstract Interpretation. pp. 266–287 (2017)
17. Hahn, E., Li, Y., Schewe, S., Turrini, A., Zhang, L.: iscasMc: A web-based probabilistic model checker. In: International Symposium on Formal Methods. pp. 312–317 (May 2014)
18. Hahn, E.M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., Wojtczak, D.: Model-free reinforcement learning for branching markov decision processes. In: Silva, A., Leino, K.R.M. (eds.) Computer Aided Verification. pp. 651–673. Springer International Publishing, Cham (2021)
19. Hahn, E.M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., Wojtczak, D.: Model-free reinforcement learning for lexicographic omega-regular objectives. In: Formal Methods - 24th International Symposium. pp. 142–159. LNCS 13047 (2021)
20. van Hasselt, H.: Double Q-learning. In: Advances in Neural Information Processing Systems. pp. 2613–2621 (2010)
21. Henzinger, T.A., Piterman, N.: Solving games without determinization. In: 15th Conference on Computer Science Logic. pp. 394–409. Szeged, Hungary (Sep 2006), LNCS 4207
22. Irpan, A.: Deep reinforcement learning doesn't work yet. https://www.alexirpan.com/2018/02/14/rl-hard.html (2018)
23. Kazemi, M., Perez, M., Somenzi, F., Soudjani, S., Trivedi, A., Velasquez, A.: Translating omega-regular specifications to average objectives for model-free reinforcement learning. In: Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems. pp. 732–741 (2022)
24. Křetínský, J., Meggendorfer, T., Sickert, S.: Owl: A library for ω-words, automata, and LTL. In: Automated Technology for Verification and Analysis, ATVA. pp. 543–550 (2018), LNCS 11138
25. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Computer Aided Verification (CAV). pp. 585–591 (Jul 2011), LNCS 6806
26. Lavaei, A., Somenzi, F., Soudjani, S., Trivedi, A., Zamani, M.: Formal controller synthesis for continuous-space mdps via model-free reinforcement learning. In: 11th ACM/IEEE International Conference on Cyber-Physical Systems,

ICCPS 2020, Sydney, Australia, April 21-25, 2020. pp. 98–107. IEEE (2020). https://doi.org/10.1109/ICCPS48487.2020.00017,
27. Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems *Specification*. Springer (1991)
28. McNaughton, R.: Testing and generating infinite sequences by a finite automaton. Inf. Control. **9**(5), 521–530 (1966)
29. Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Human-level control through deep reinforcement learning. Nature **518** (2015)
30. Naik, A., Shariff, R., Yasui, N., Sutton, R.S.: Discounted reinforcement learning is not an optimization problem. CoRR **abs/1910.02140** (2019), http://arxiv.org/abs/1910.02140
31. OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., Zhang, L.: Solving rubik's cube with a robot hand. arXiv preprint (2019)
32. Perrin, D., Pin, J.É.: Infinite Words: Automata, Semigroups, Logic and Games. Elsevier (2004)
33. Pnueli, A.: The temporal semantics of concurrent programs. Theoret. Comput. Science **13**, 45–60 (1981)
34. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc., New York, NY, USA (1994)
35. Sadigh, D., Kim, E., Coogan, S., Sastry, S.S., Seshia, S.A.: A learning based approach to control synthesis of Markov decision processes for linear temporal logic specifications. In: IEEE Conference on Decision and Control (CDC). pp. 1091–1096 (Dec 2014)
36. Schewe, S., Tang, Q., Zhanabekova, T.: Deciding what is good-for-mdps. CoRR **abs/2202.07629** (2022), https://arxiv.org/abs/2202.07629
37. Sickert, S., Esparza, J., Jaax, S., Křetínský, J.: Limit-deterministic Büchi automata for linear temporal logic. In: Computer Aided Verification (CAV). pp. 312–332 (2016), LNCS 9780
38. Silver, D., et al.: Mastering the game of Go with deep neural networks and tree search. Nature **529**, 484–489 (Jan 2016)
39. Simovec, P.: Transformation of nondeterministic büchi automata to slim automata (2021), https://is.muni.cz/th/nd15g/
40. Sutton, R.S.: Learning to predict by the method of temporal differences. Machine Learning **3**, 9–44 (1998)
41. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, second edn. (2018)
42. Wan, Y., Naik, A., Sutton, R.S.: Learning and planning in average-reward markov decision processes. In: International Conference on Machine Learning. pp. 10653–10662. PMLR (2021)
43. Watkins, C.J.C.H., Dayan, P.: Q-learning. In: Machine Learning. pp. 279–292 (1992)
44. Wiewiora, E.: Reward shaping. In: Encyclopedia of Machine Learning, pp. 863–865. Springer (2010)
45. Yang, C., Littman, M., Carbin, M.: Reinforcement learning for general ltl objectives is intractable. arXiv preprint arXiv:2111.12679 (2021)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.