# COOL-MC: A Comprehensive Tool for Reinforcement Learning and Model Checking

Dennis Gross[1(✉)], Nils Jansen[1], Sebastian Junges[1], and Guillermo A. Pérez[2]

[1] Radboud University, Nijmegen, The Netherlands
dgross@science.ru.nl
[2] University of Antwerp – Flanders Make, Antwerp, Belgium

**Abstract.** This paper presents COOL-MC, a tool that integrates state-of-the-art reinforcement learning (RL) and model checking. Specifically, the tool builds upon the OpenAI gym and the probabilistic model checker Storm. COOL-MC provides the following features: (1) a simulator to train RL policies in the OpenAI gym for Markov decision processes (MDPs) that are defined as input for Storm, (2) a new model builder for Storm, which uses callback functions to verify (neural network) RL policies, (3) formal abstractions that relate models and policies specified in OpenAI gym or Storm, and (4) algorithms to obtain bounds on the performance of so-called permissive policies. We describe the components and architecture of COOL-MC and demonstrate its features on multiple benchmark environments.

## 1 Introduction

Deep Reinforcement learning (RL) has created a seismic shift in how we think about building agents for dynamic systems [32,33]. It has triggered applications in critical domains like energy, transportation, and defense [4,13,34]. An *RL agent* aims to learn a near-optimal policy regarding some fixed objective by taking actions and perceiving feedback signals, usually rewards and observations of the *environment* [36]. Unfortunately, learned policies come with no guarantee to avoid *unsafe behavior* [14]. Generally, rewards lack the expressiveness to encode complex safety requirements [37] and it is hard to assess whether the training at a certain point in time is sufficient.

To resolve the issues mentioned above, verification methods like model checking [3,8] are used to reason about the safety of RL, see for instance [5,18,23,38]. However, despite the progress in combining these research areas, there is—to the best of our knowledge—no mature tool support that tightly integrates exact model checking of state-of-the-art deep RL policies. One of the reasons is that policies obtained on an OpenAI environment may be incompatible with a related

Download it from https://lava-lab.github.io/COOL-MC/.


W. Dong and J.-P. Talpin (Eds.): SETTA 2022, LNCS 13649, pp. 41–49, 2022.
https://doi.org/10.1007/978-3-031-21213-0_3

formal model (that can be used for model checking), and vice versa. These incompatibilities are often due to differences in their state or action spaces, or even their rewards. Another challenge is that verifying deep RL policies currently requires different algorithms, data structures, and abstractions depending on the architecture and size of a neural network (NN).

We present COOL-MC, an open-source tool that integrates the OpenAI gym with the probabilistic model checker Storm [11]. The purpose of the tool is *to enable training RL policies while being able to verify them at any stage of the training process*. Concretely, we focus on supporting the following methodology as main use case for COOL-MC. An RL expert is tasked with training an RL policy with formal guarantees. These guarantees are captured as formal specifications that hold on a well-defined formal model, formulated by the expert. At any stage of the learning, the RL expert wants to establish whether the policy has the desired properties as given by the formal model and specification.

Thus, the input of COOL-MC consists of two models of the environment: (1) an *OpenAI-gym compatible environment*, to train an RL policy; (2) an *Markov decision process (MDP)*, specified using the PRISM language [31], to verify the policy together with a formal specification e.g., a probabilistic computation tree logic (PCTL) formula. Only the MDP model of the environment is required: If no OpenAI-gym environment is given, COOL-MC provides a wrapper to cast the MDP as an OpenAI gym environment. The latter is done using a *syntax-based simulator* that avoids building the MDP explicitly in Storm. Training of an RL policy can be done using any RL agent available in the OpenAI gym. Any (trained) policy can then be formally verified via Storm using *callback functions* that query the NN and build the induced discrete-time Markov chain (DTMC) incrementally [7,10]. COOL-MC assumes no formal relation between the two given environment models. Of course, for the purpose of policy verification, the states of both have to be in some of relation. For this purpose, we support abstraction of models and policies. In particular, we employ a feature-based representation for MDPs [35]. Such features may, for instance, refer to concrete positions of agents in their environment, or to the fuel level of a car. We offer the user the option to *remap feature values* and to define *abstractions* of their feature domains [24] (see Sect. 2). This yields the opportunity to verify these formal abstraction and obtaining bounds on the actual policy.

*Related Work.* The recently developed MoGym [15] is built on top of the MODEST toolset [20], and it is related to our tool. The main difference is that our tool supports so-called permissive policies and feature remappings. Likewise, Bacci et al. tackle the problem of verifying stochastic RL policies for continuous environments via an abstraction approach [2] but they abstract the policy using mixed-integer LPs. This limits the NN architectures their tool can handle. In contrast, we take the trained policy exactly as is and are architecture-agnostic. Gu et al. propose a method called MCRL, that combines model checking with reinforcement learning in a multi-agent setting for mission planning to ensure that safety-critical requirements are met [17]. The difference in our approach is that we allow the post-verification of single RL policies in every kind of environment that can be modeled in the PRISM language. Mungojerrie uses RL to obtain an

optimal policy w.r.t. a given $\omega$-regular objective [19]. In [23], a similar approach is presented for objectives given in linear temporal logic (LTL). Shielded RL guides the RL agent during training to avoid safety violations [1,9,10,25]. We, on the other hand, do not guide the training process but verify the trained policy.

## 2 Core Functionality and Architecture of COOL-MC

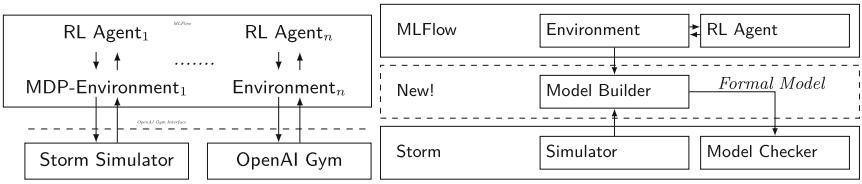
We now present the core functionality and the architecture of COOL-MC.

*Training.* During RL training, interaction with the user-provided environment takes place as follows. Starting from the current state, the agent selects an action upon which the environment returns the next state and the corresponding reward to derive a near-optimal policy.

*Verification.* To allow model checking the trained policy (an RL agent) against a user-provided specification and formal model, we build the induced DTMC on the fly via Storm callback functions. For every reachable state by the policy, the policy is queried for an action. In the underlying MDP, only states that may be reached via that action are expanded. The resulting model is fully probabilistic, as no action choices are left open. It is, in fact, the Markov chain induced by the original MDP and the policy.

*Policy Transformations.* We extend the above-described on-the-fly construction with two abstractions. One of them relies on feature abstraction [12,28], while the other uses a coarser abstraction of the state variables [24]. In both cases, we obtain a new policy $\tau$.

**1. Feature abstraction:** Assume the state space has some structure $S \subseteq Q \times I$ for suitable $Q$ and $I$. Thus, states are pairs $s = (q, i)$ where $q$ and $i$ correspond to *features* whose values range over $Q$ and $I$ respectively. Let $\mathsf{Act}$ denote the actions in the MDP. Given a partition $K_1, \ldots, K_n$ of $I$, we abstract this feature from the policy $\pi$, by defining a *permissive* policy [12] $\tau \colon S \to 2^{\mathsf{Act}}$, i.e., a policy selecting multiple actions in every state. In particular, we consider $\tau(q, i) = \bigcup_{k \in K_n} \pi(q, k)$, with $K_n$ is the unique set such that $i \in K_n$. This policy $\tau$ ignores the value of $i$ in state $(q, i)$ and instead selects any action that can be selected from states $(q, k)$, with $k \in K_n$. Applying a permissive policy yields an induced MDP, which can be model checked to provide best- and worst-case bounds.
**2. Feature remapping:** In this case, we again assume $S \subseteq Q \times I$. Given a mapping $\mu : I \to Y$ with $Y \subseteq I$, we define the abstracted policy $\tau$ as follows. $\tau(q, i) = \pi(q, \mu(i))$ The feature values of $i$ are effectively being transformed into values from a (possibly smaller) set of feature values $Y \subseteq I$ before being fed into the original policy.




(a) RL Training. (b) RL Agent Verification.

**Fig. 1.** COOL-MC architecture.

*Architecture.* COOL-MC is a Python toolchain on top of Storm and OpenAI gym. It also employs MLflow[1] as a file manager, which manages all the user experiments in the local file system, and allows the user to compare different experiments via the MLflow webserver. Furthermore, each task (for example, training or verification) is a separate MLflow component in the software architecture of COOL-MC and is controlled by the main script. COOL-MC also contains the *RL agent* component, which is a wrapper around the policy and interacts with the *environment*. Currently implemented agents include Q-Learning [39], Hillclimbing [30], Deep Q-Learning, and REINFORCE [40]. For training uses the Storm simulator or an OpenAI Gym (Fig. 1a). For verification, the model builder creates an induced DTMC which is then checked by Storm (Fig. 1b).

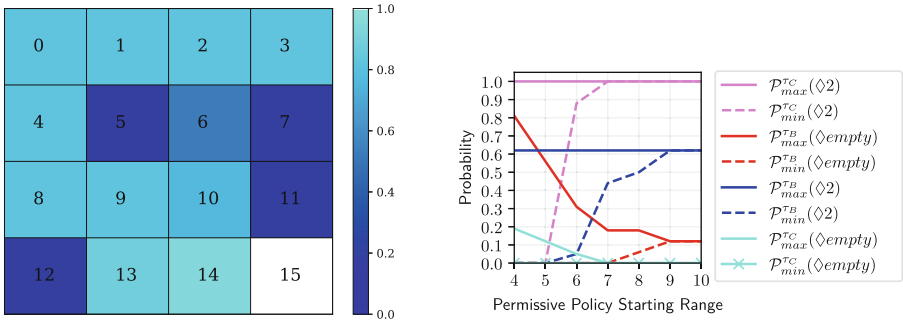
## 3 Numerical Experiments

Our experimental setup allows us to showcase the core features of COOL-MC. For clarity of exposition, in this paper, we focus on describing concise examples: In the *Frozen lake*, the agent has to reach a Frisbee on a frozen lake. At every step, the agent can choose to move "up", "down", "left", or "right". The execution of said movement is imprecise because of the ice: it is only as intended in 33.33% of the cases. In the remaining 66.66% of the cases, another movement is executed—the only restriction is that the agent cannot move in the direction opposite than its choice. The agent receives a reward of +1 if it reaches the Frisbee [6]. In the *Taxi* environment, the agent has to transport customers to their destination without running out of fuel; in the *Collision Avoidance* environment, it must avoid two obstacles in a 2D grid; the descriptions of the other environments (*Smart Grid, Stock Market, Atari James Bond, Atari Crazy Climber*)[2] can be found in the Appendix of our technical report [16]. We stress that the tool can handle several other benchmarks, e.g., PRISM-format MDPs from the *Quantitative Verification Benchmark Set* [21]. All experiments were executed on

[1] MLflow is a platform to streamline ML development, including tracking experiments, packaging code into reproducible experiments, and sharing and deploying models [41].

[2] We refer the interested reader to the repository https://github.com/LAVA-LAB/COOL-MC of the tool for more experiments with these and other environments.

a laptop with 8 GB RAM and an Intel(R) Core(TM) i7-8750H CPU@2.20 GHz × 12.




(a) Heatmap shows per state $s$ the value $\mathcal{P}^{s,\pi_A}(\Diamond frisbee)$.

(b) Plot shows different permissive taxi policies "lumping" fuel levels.

**Fig. 2.** Frozen lake verification (a) and permissive taxi policies (b).

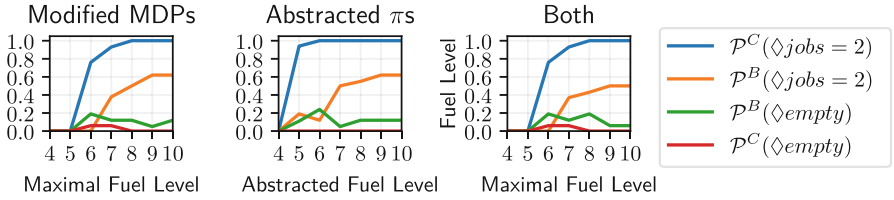



**Fig. 3.** Plots for the probabilities of running out of fuel on the road and finishing two jobs in three different settings for the trained policies $\pi_B$ and $\pi_C$.

*Policy Verification.* The frozen lake environment is a commonly used OpenAI gym benchmark. Therefore, we trained a deep-neural-network policy $\pi_A$ in this OpenAI gym for $100K$ episodes. For more information on the training process, see Appendix of our technical report [16]. After the learning process, we verified $\pi_A$ in the frozen lake PRISM model (which describes the OpenAI environment). We investigate if the trained policy ultimately learned to account for the slippery factor. Figure 2a shows that the agent only falls into the water at the fifth position (from state 6 while selecting the action "left"). The agent reaches state 9 with a probability of $\mathcal{P}^{0,\pi_A}(\Diamond 9) = 1$. This indicates that the trained RL agent has learned to take the slippery factor into account since otherwise, it would not safely reach state 9. At state 9, we split the environment into an area in which the RL agent cannot fall into the water by following its policy and an unsafe

area. If the agent successfully selects the action "down", it stays safe the rest of the trajectory to the Frisbee. Suppose the agent slips to state 10 while selecting "down", the probability of reaching the Frisbee declines to $\mathcal{P}^{10,\pi_A}(\Diamond frisbee) = 0.76$.

*Scalability and Performance Analysis.* A $10 \times 10$ instance of the collision avoidance benchmark with a slickness constant value of 0.1 results in an MDP with $1,077,628$ states and $118,595,768$ transitions. This already causes Storm to run out of memory. If we apply a policy (trained with COOL-MC), the induced DTMC has $1M$ states and $29,256,683$ transitions, and is now within reach for Storm, while the result may not be optimal. COOL-MC can handle sizes of up to $11 \times 11$ and a slickness assignment of 0.1 without running out of memory (see Table 1). The bottleneck of our tool is the model building time (see Table 1). Model checking times are negligible in comparison.

**Table 1.** Benchmarks for different environments and trained policies. Times are measured in seconds; *OOM* stands for out of memory.

| Environment | MDP | | DTMC | | | | | |
|---|---|---|---|---|---|---|---|---|
| | States | Trans. | Specification | Result | Build time | Check. time | States | Trans. |
| Frozen lake | 17 | 152 | $\mathcal{P}^{\pi A}(\Diamond water)$ | 0.18 | 0.25 | 0 | 14 | 34 |
| Taxi | 8576 | 39608 | $\mathcal{P}^{\pi C}(\Diamond empty)$ | 0 | 4.78 | 0 | 252 | 507 |
| Taxi | 8576 | 39608 | $\mathcal{P}^{\pi C}(\Diamond 2)$ | 1 | 4.5 | 0 | 252 | 507 |
| Collision $10 \times 10$ | 1077628 | 118595768 | $\mathcal{T}^{\pi D}(\Diamond collide)$ | 470.73 | 3106 | 226 | 1000000 | 29256683 |
| Collision $11 \times 11$ | 1885813 | 211956692 | $\mathcal{T}^{\pi D}(\Diamond collide)$ | 546.18 | 3909 | 314 | 1771561 | 52433826 |
| Collision $12 \times 12$ | 3148444 | 359797152 | $\mathcal{T}^{\pi D}(\Diamond collide)$ | OOM | 6619 | OOM | 2985984 | 89198366 |
| Smart grid | 271 | 10144 | $\mathcal{P}^{\pi E}_{\leq 1000}(\Diamond black)$ | 0.02 | 0.48 | 0 | 40 | 176 |
| Stock market | 14760 | 89506 | $\mathcal{P}^{\pi F}(\Diamond loss)$ | 0 | 0.3 | 0 | 130 | 377 |
| James bond | 172032 | 3182592 | $\mathcal{P}^{\pi G}_{\leq 15}(\Diamond done)$ | 0.23 | 1997 | 0.28 | 159744 | 1105920 |
| Crazy climber | 57344 | 499712 | $\mathcal{P}^{\pi G}_{\leq 15}(\Diamond done)$ | 0 | 175 | 0 | 8193 | 32772 |

*Feature Remapping.* The policies $\pi_B$ and $\pi_C$ are trained in the taxi environment to transport passengers to their destinations. We are now interested in what happens if the taxi policies are deployed in cars with different maximal fuel-tank capacities. We can measure how well they perform for different such capacities using COOL-MC (left plot, Fig. 3). On the other hand, to reduce design or hardware costs, one might be interested in deploying a single policy with a "virtual" or abstracted maximal fuel-tank capacity. Feature remapping can obtain such policies from $\pi_B$ or $\pi_C$. For example, an abstract fuel level of 6 means that all fuel levels 6–10 will be perceived as fuel level 6. The modified policy can then be evaluated in the original MDP (middle plot, Fig. 3). Perhaps more interestingly, one can choose a particular abstraction and evaluate its performance at different physical max fuel-tank capacities (right plot, Fig. 3).

*Feature Abstraction.* In Fig. 2b, we first transform the trained policies $\pi_B, \pi_C$ into permissive ones $\tau_B, \tau_C$ due to, for instance, the lack of exact fuel-level sensors in the deployment hardware. To counter this loss of precision, we can get a best- and worst-case analysis under $\tau_B, \tau_C$ for different fuel-level ranges. For example, a starting range of 8 means that COOL-MC creates a permissive policy that "lumps" fuel-levels 8, 9, and 10 together. Permissive policies with larger starting ranges have min/max probabilities closer together.

## 4 Conclusion and Future Work

We presented the tool COOL-MC, which provides a tight interaction between model checking and reinforcement learning. In the future, we will extend the tool to directly incorporate safe reinforcement learning approaches [22,25–27] and will extend the model expressivity to partially observable MDPs [29].

## References

1. Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., Topcu, U.: Safe reinforcement learning via shielding. In: AAAI, pp. 2669–2678. AAAI Press (2018)
2. Bacci, E., Parker, D.: Verified probabilistic policies for deep reinforcement learning. CoRR abs/2201.03698 (2022)
3. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge (2008)
4. Boron, J., Darken, C.: Developing combat behavior through reinforcement learning in wargames and simulations. In: CoG, pp. 728–731. IEEE (2020)
5. Brázdil, T., et al.: Verification of Markov decision processes using learning algorithms. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 98–114. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11936-6_8
6. Brockman, G., et al.: OpenAI gym. CoRR abs/1606.01540 (2016)
7. Cassez, F., David, A., Fleury, E., Larsen, K.G., Lime, D.: Efficient on-the-fly algorithms for the analysis of timed games. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 66–80. Springer, Heidelberg (2005). https://doi.org/10.1007/11539452_9
8. Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.): Handbook of Model Checking. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8
9. David, A., et al.: On time with minimal expected cost! In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 129–145. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11936-6_10
10. David, A., Jensen, P.G., Larsen, K.G., Mikučionis, M., Taankvist, J.H.: Uppaal Stratego. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 206–211. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_16
11. Dehnert, C., Junges, S., Katoen, J.-P., Volk, M.: A Storm is coming: a modern probabilistic model checker. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 592–600. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_31
12. Dräger, K., Forejt, V., Kwiatkowska, M.Z., Parker, D., Ujma, M.: Permissive controller synthesis for probabilistic systems. Log. Methods Comput. Sci. **11**(2) (2015)

13. Farazi, N.P., Zou, B., Ahamed, T., Barua, L.: Deep reinforcement learning in transportation research: a review. Transp. Res. Interdisc. Perspect. **11**, 100425 (2021)
14. García, J., Fernández, F.: A comprehensive survey on safe reinforcement learning. J. Mach. Learn. Res. **16**, 1437–1480 (2015)
15. Gros, T.P., Hermanns, H., Hoffmann, J., Klauck, M., Köhl, M.A., Wolf, V.: MoGym: using formal models for training and verifying decision-making agents. In: Shoham, S., Vizel, Y. (eds.) CAV 2022. LNCS, vol. 13372, pp. 430–443. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-13188-2_21
16. Gross, D., Jansen, N., Junges, S., Perez, G.A.: COOL-MC: a comprehensive tool for reinforcement learning and model checking. arXiv preprint arXiv:2209.07133 (2022)
17. Gu, R., Jensen, P.G., Poulsen, D.B., Seceleanu, C., Enoiu, E., Lundqvist, K.: Verifiable strategy synthesis for multiple autonomous agents: a scalable approach. Int. J. Softw. Tools Technol. Transfer **24**, 395–414 (2022). https://doi.org/10.1007/s10009-022-00657-z
18. Hahn, E.M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., Wojtczak, D.: Omega-regular objectives in model-free reinforcement learning. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11427, pp. 395–412. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17462-0_27
19. Hahn, E.M., Perez, M., Schewe, S., Somenzi, F., Trivedi, A., Wojtczak, D.: Mungojerrie: reinforcement learning of linear-time objectives. CoRR abs/2106.09161 (2021)
20. Hartmanns, A., Hermanns, H.: The modest toolset: an integrated environment for quantitative modelling and verification. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 593–598. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_51
21. Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E.: The quantitative verification benchmark set. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11427, pp. 344–350. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17462-0_20
22. Hasanbeig, M., Kroening, D., Abate, A.: Towards verifiable and safe model-free reinforcement learning. In: OVERLAY@AI*IA. CEUR WS, vol. 2509, p. 1. CEUR-WS.org (2019)
23. Hasanbeig, M., Kroening, D., Abate, A.: Deep reinforcement learning with temporal logics. In: Bertrand, N., Jansen, N. (eds.) FORMATS 2020. LNCS, vol. 12288, pp. 1–22. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-57628-8_1
24. Jaeger, M., Jensen, P.G., Guldstrand Larsen, K., Legay, A., Sedwards, S., Taankvist, J.H.: Teaching stratego to play ball: optimal synthesis for continuous space MDPs. In: Chen, Y.-F., Cheng, C.-H., Esparza, J. (eds.) ATVA 2019. LNCS, vol. 11781, pp. 81–97. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-31784-3_5
25. Jansen, N., Könighofer, B., Junges, S., Serban, A., Bloem, R.: Safe reinforcement learning using probabilistic shields (invited paper). In: CONCUR. LIPIcs, vol. 171, pp. 3:1–3:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
26. Jin, P., Tian, J., Zhi, D., Wen, X., Zhang, M.: Trainify: a CEGAR-driven training and verification framework for safe deep reinforcement learning. In: Shoham, S., Vizel, Y. (eds.) CAV 2022. LNCS, vol. 13371, pp. 193–218. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-13185-1_10
27. Jothimurugan, K., Bansal, S., Bastani, O., Alur, R.: Specification-guided learning of nash equilibria with high social welfare. In: Shoham, S., Vizel, Y. (eds.) CAV

2022. LNCS, vol. 13372, pp. 343–363. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-13188-2_17
28. Junges, S., Jansen, N., Dehnert, C., Topcu, U., Katoen, J.-P.: Safety-constrained reinforcement learning for MDPs. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 130–146. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_8
29. Junges, S., Jansen, N., Seshia, S.A.: Enforcing almost-sure reachability in POMDPs. In: Silva, A., Leino, K.R.M. (eds.) CAV 2021. LNCS, vol. 12760, pp. 602–625. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81688-9_28
30. Kimura, H., Yamamura, M., Kobayashi, S.: Reinforcement learning by stochastic hill climbing on discounted reward. In: ICML, pp. 295–303. Morgan Kaufmann (1995)
31. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 2.0: a tool for probabilistic model checking. In: QEST, pp. 322–323. IEEE Computer Society (2004)
32. Levine, S., Finn, C., Darrell, T., Abbeel, P.: End-to-end training of deep visuomotor policies. J. Mach. Learn. Res. **17**, 39:1–39:40 (2016)
33. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
34. Nakabi, T.A., Toivanen, P.: Deep reinforcement learning for energy management in a microgrid with flexible demand. Sustain. Energy Grids Netw. **25**, 100413 (2021)
35. Strehl, A.L., Diuk, C., Littman, M.L.: Efficient structure learning in factored-state MDPs. In: AAAI, pp. 645–650. AAAI Press (2007)
36. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (2018)
37. Vamplew, P., et al.: Scalar reward is not enough: a response to Silver, Singh, Precup and Sutton (2021). Auton. Agents Multi Agent Syst. **36**(2) (2022). Article number: 41. https://doi.org/10.1007/s10458-022-09575-5
38. Wang, Y., Roohi, N., West, M., Viswanathan, M., Dullerud, G.E.: Statistically model checking PCTL specifications on Markov decision processes via reinforcement learning. In: CDC, pp. 1392–1397. IEEE (2020)
39. Watkins, C.J., Dayan, P.: Q-learning. Mach. Learn. **8**(3–4), 279–292 (1992). https://doi.org/10.1007/BF00992698
40. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. **8**, 229–256 (1992). https://doi.org/10.1007/BF00992696
41. Zaharia, M., et al.: Accelerating the machine learning lifecycle with MLflow. IEEE Data Eng. Bull. **41**(4), 39–45 (2018)