Validation of Reinforcement Learning Agents and Safety Shields with ProB

Fabian Vu(✉), Jannik Dunkelau(✉), and Michael Leuschel

Mathematisch-Naturwissenschaftliche Fakultät Institut für Informatik, Heinrich-Heine-Universität Düsseldorf, Düsseldorf, Germany
{fabian.vu,jannik.dunkelau,leuschel}@uni-duesseldorf.de

Abstract. Reinforcement learning (RL) is an important machine learning technique to train agents that make decisions autonomously. For safety-critical applications, however, the decision-making of an RL agent may not be intelligible to humans and thus difficult to validate, verify and certify.

This work presents a technique to link a concrete RL agent with a high-level formal B model of the safety shield and the environment. This allows us to run the RL agent in the formal method tool ProB, and particularly use the formal model to surround the agent with a safety shield at runtime. This paper also presents a methodology to validate the behavior of RL agents and respective safety shields with formal methods techniques, including trace replay, simulation, and statistical validation. The validation process is supported by domain-specific visualizations to ease human validation. Finally, we demonstrate the approach for a highway simulation.

Keywords: AI · Reinforcement Learning · B Method · Validation · Shielding

1 Introduction and Motivation

Artificial intelligence (AI) and machine learning (ML) [42] are increasingly used to develop software applications. One popular ML technique is reinforcement learning (RL) [31] which also finds use in safety-critical domains such as the automotive domain [27], the railway domain [21], and the aviation domain [24]. Hereby, an *agent* learns to make autonomous decisions within an *environment* to maximize an accumulated *reward*. In a trial-and-error approach, the agent receives a reward as feedback for actions taken based on their observed outcome and uses this feedback to optimize its *decision policy*.

The work of Fabian Vu is part of the KI-LOK project funded by the "Bundesministerium für Wirtschaft und Energie"; grant # 19/21007E, and the IVOIRE project funded by "Deutsche Forschungsgemeinschaft" (DFG) and the Austrian Science Fund (FWF) grant # I 4744-N.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
N. Benz et al. (Eds.): NFM 2024, LNCS 14627, pp. 279–297, 2024.
https://doi.org/10.1007/978-3-031-60698-4_16

In the context of safety-critical applications, it is important to *verify* and *validate* an RL agent's learned behavior. As RL agents typically are black boxes, their decision-making may be unintelligible and hard to reason about. Validation and verification of RL agents is thus an ongoing research topic [6,14,34]. *Safety shields* [4] is a runtime monitoring and verification technique to ensure the safety of RL agents. A safety shield intervenes when a dangerous situation might occur, i.e., its task is to avoid or prevent dangerous situations. Safety shields are related to Sha's concept of "using simplicity to control complexity" [29] where a simpler system monitors and intervenes in a complex system when rules are violated.

This work presents a technique to link a concrete RL agent with a high-level formal model of the B method [2] for the RL agent and its environment with a safety shield. This allows us to run the RL agent in the ProB [16,17] animator and model checker, and use the formal model as a safety shield at runtime. While ProB also supports verification of the formal model via model checking, we focus on the validation of RL agents with other formal methods techniques such as trace replay, simulation, and statistical validation. With trace replay, it is possible to re-play a single execution run to reason about the RL agent's decisions. Trace replay also checks whether an execution run is feasible; thus, one can validate whether a safety shield has out-ruled a dangerous situation. Using SimB [38], one can run the RL agent in ProB in real-time, or as Monte Carlo simulation. Based on multiple simulated runs, one can apply statistical validation such as computing the likelihood of violating certain properties, and estimating probabilities, averages, and sums. Finally, we demonstrate the applicability and efficacy of this methodology in a highway environment [15]. In this context, we evaluate how safety shields in this work improve the safety and the achieved reward for the RL agent. We also use the insights gained from this technique, to improve the safety shield and the reward function.

2 Background

The B Method. The B method [2] is a formal method for specifying and verifying software systems. The B language is based on set theory and first-order logic, and makes use of *general substitution* for state modifications as well as *refinement calculus* to model *state machines* at various levels of abstraction.

Within a B model, the modeler has to specify an `INVARIANT` clause which contains a predicate to provide typing for variables and define (safety) properties which must be fulfilled in each state of the model. The `INITIALISATION` contains substitutions (also

Listing 1. B Model for Coin Toss

```
1 MACHINE CoinToss
2 SETS Side = {Heads, Tails, None}
3 VARIABLES lastToss
4 INVARIANT lastToss ∈ Side
5 INITIALISATION lastToss := None
6 OPERATIONS
7   toss = lastToss :∈ {Heads, Tails}
8 END
```

called statements) to describe the model's initial states, assigning values to each machine variable. Within the `OPERATIONS` clause, a modeler can specify operations with respective *guards* and substitutions. When the guard is true, the operation's substitution can be executed by modifying the model's current state. Listing 1 shows a simple B model for a coin toss with an operation `toss` that chooses between `Heads` and `Tails` non-deterministically.

In this paper, we use established tools from the B landscape, namely ProB and SimB. ProB [16,17] is an animator, constraint solver, and model checker for formalisms such as B, Event-B, TLA$^+$ or CSP. It provides capabilities such as animation, trace replay [5], simulation [38], and different model checking techniques [13,23] to *verify* and *validate* formal models. SimB [37,38] is a simulator with support for timing, probabilities, and live user interaction. SimB also provides statistical validation techniques such as hypothesis testing and value estimation for probabilities, averages, and sums.

Reinforcement Learning. Reinforcement Learning [31] is a machine learning paradigm in which an agent learns to maximize a cumulative reward function via a feedback loop with its *environment* in a trial-and-error manner. The agent interacts with its environment through a set of available actions which can alter the environment's state. The respective actions are chosen via a gradually learned *policy* which dictates the agent's decision-making process. The benefits of actions are quantified by a *reward* function evaluated in the successor states. By estimating the *value* (i.e. predicted long-term reward) of actions, instead of only the immediate reward of the next state, a policy can make short-term trade-offs which lead to higher long-term rewards.

In this work, we use the Deep Q-Network (DQN) algorithm [20] which mixes deep learning with Q learning [40]. Given a state-action pair (s, a), the idea behind Q learning revolves around learning an action-value function $Q(s, a)$ that estimates the long-term value of executing action a in state s [31]. In the DQN algorithm, the learning of the Q function is done by a deep neural network [20].

Safety Shields. Safety shields [4] is a formal technique to ensure the safety of an agent at runtime. More precisely, the agent is surrounded by a *safety shield* which intervenes to prevent/avoid dangerous actions. Safety shields align with Sha's concept of "using simplicity to control complexity" [29] where a simpler system monitors and enforces properties/rules in a complex system. Two techniques are pre-shielding and post-shielding [12]. In the pre-shielding approach, actions are shielded before execution and then provided to an RL agent to choose the next action from. In post-shielding, actions are corrected to safe ones when the agent's decisions are considered unsafe. In shield synthesis [12] the safety shield is synthesized via training from the underlying environment and RL agent.

The Highway Environment. The highway environment [15] is an available environment for training RL agents to navigate a particular vehicle on a highway. We refer to that vehicle as the *ego vehicle*. The observed environment contains positional information (x/y-coordinate) and velocities (in x/y direction) of all

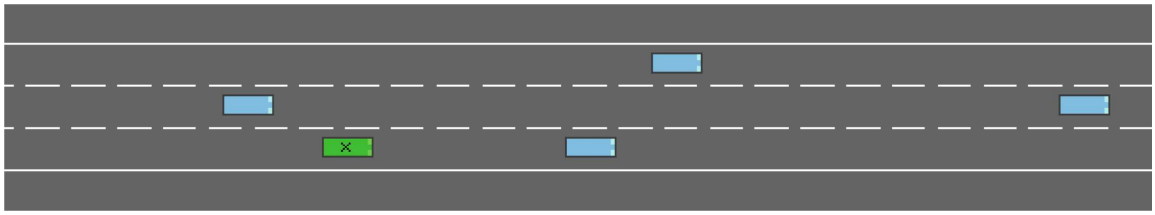


Fig. 1. Screenshot from the highway environment. The green box (manually marked with an X, on the bottom line to the left) represents the ego vehicle controlled by the RL agent, while the blue boxes are surrounding vehicles. (Color figure online)

vehicles. There is information about whether the ego vehicle has crashed, and the reward resulting from the current state. Hereby, the reward function favors driving fast and on the right-most lane. The environment is simulated in a frequency of one frame per second, following the default configuration. Hence, each second the RL agent observes the current state and reacts accordingly. As the goal is to learn a policy which lets the ego vehicle drive fast and collision-free, the agent has to learn when to accelerate or decelerate, and when to switch lanes to keep momentum. The agent's action space consists of 5 actions: `IDLE`, `LANE_LEFT`, `LANE_RIGHT`, `FASTER`, and `SLOWER`. Figure 1 displays a visualization of an exemplary environment state.

3 Formal Models for Reinforcement Learning Agents

This section presents a technique to use formal models for the validation of RL agents. Based on a trained RL agent, a modeler creates a formal model which captures the RL agent's actions/decisions, and the environment's state. In this work, we do not formally model the internal decision-making process of the RL agent. This means that the decisions are still made by the RL agent, while its decision and the resulting environment's state are synchronized with the formal model. Adding the agent's actions as machine operations in the formal model, we can also define rules in the formal model to use it as a safety shield (discussed in detail in Sect. 3.2).

Figure 2 illustrates the interaction between the formal model and the RL agent with Shielding. During the RL agent's runtime, there is a sensor capturing the RL agent's environment. The environment is updated in the formal model and the RL agent accordingly. At runtime, the formal model is used to compute the set of actions that are considered to be safe, which is then passed to the RL agent. From these safe actions, the RL agent then chooses the one with the highest estimated long-term reward. This action is then executed in the environment.

Using a formal model at runtime gives us the ability to apply formal method tools and techniques to the RL agent. This enables us to evaluate and uncover weaknesses in the reward function and (possible) safety shields. Furthermore, this work is not limited to RL, but caters to other AI and real-time systems.

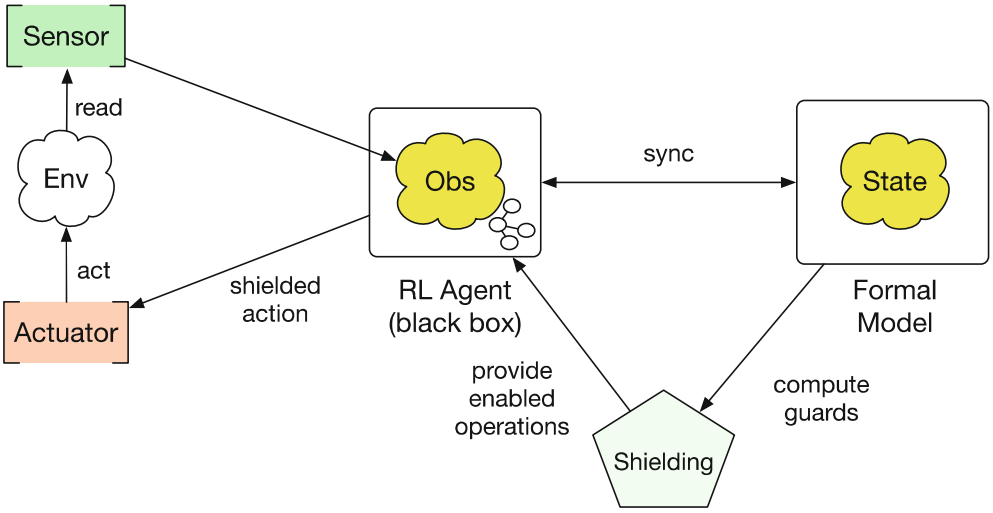


Fig. 2. Interaction between Formal Model, RL Agent with Shielding, and the Environment; shielding process works similarly to pre-shielding [12].

3.1 Creation of the Formal Model

The formal B model contains (1) the current state of the environment, and (2) the agent's actions. The environment's current state is represented using *sets*, *constants*, and *variables* in the formal model.

Let us assume that the RL agent can execute the actions $a_1, \ldots, a_m$. For each action a_j with $j \in \{1, \ldots, m\}$, we introduce a respective operation o_j which consists of a guard g_{o_j} and a state-altering substitution s_{o_j}:

$$o_j = \texttt{PRE}\ g_{o_j}\ \texttt{THEN}\ s_{o_j}\ \texttt{END}$$

Each operation's guard g_{o_j} defines whether the operation is considered safe for execution; we use this to encode a safety shield. The guards must hence be encoded in such a way that at least one operation is always enabled. Otherwise, the agent runs at risk of being unable to act at all in certain cases, as it will only be able to execute actions with their corresponding guards enabled. This property can be checked by techniques that are made available in this work.[1] Within the substitution s_{o_j}, the variables are assigned to a possible value wrt. their expected domain. The `INITIALISATION` substitution is encoded similarly. With this encoding, it is also possible to validate the implementation of the RL agent. Let us assume that v_i is a variable whose value changes after executing an action o_j. Within s_{o_j}, one could then encode:

- an assignment by value val ($v_i := val$),
- a non-deterministic assignment via a domain set S ($v_i :\in S$), or
- a non-deterministic assignment via a domain predicate P ($v_i :| (P)$).

Let us assume that we create a formal B model for the highway environment in Sect. 2 using a variable `velocity`. Assume we would like to encode a `FASTER`

[1] Cf. relative deadlock freedom [1, Chapter 14] for a proof-based approach.

operation with the following conditions: (1) `FASTER` shall only be executable if the `velocity` is less than or equal to 30 m/s, and (2) the `velocity` is expected to increase when executing `FASTER`. We could then encode this by an operation:

$$\texttt{FASTER} = \texttt{PRE velocity} \leq 30 \\ \texttt{THEN velocity} :| (\texttt{velocity} > \texttt{velocity}') \texttt{ END}$$

Remark: $\texttt{velocity}'$ refers to the previous state; thus, $\texttt{velocity} > \texttt{velocity}'$ means that the speed increases after the action has been executed.

3.2 Implementing a Safety Shield around the RL Agent

Referring to Fig. 2, we implemented the synchronization and communication (including shielding) between the formal model and the RL agent in ProB and SimB. The simulation is done by the RL agent and synchronized with the simulation in ProB and SimB. As mentioned before, the formal B model encodes safety shields in the operations' guards to apply pre-shielding [12]. The decision process with shielding is illustrated in Fig. 3. For each executed action, the following steps are performed:

1. The current state of the environment and the last executed action is captured by the RL agent, and provided to ProB.
2. ProB synchronizes the internal state of the animated formal model to match the current observation provided by the environment. Based on the encoding of the operations (discussed in Sect. 3.1), ProB also checks that the target state matches the desired effect of the provided action.
3. Based on the current state, ProB computes enabled operations by evaluating their guards. Actions where the guard is violated in the current state are deemed unsafe.
4. ProB provides a list of enabled operations to the RL agent.[2]
5. Based on the current observation, the RL agent predicts the enabled operation/action with the highest reward.
6. The chosen action is subsequently executed by an actuator.
7. The environment changes according to the action and the respective reward is computed.

Referring to the highway environment in Sect. 2, an example of the shielding process in Fig. 3 could be as follows: First, the RL agent observes the environment containing other vehicles and provides the information to ProB. Second, ProB computes `SLOWER` and `LANE_LEFT` as operations that are considered to be safe and provides the shielded actions to the RL agent. Finally, the RL agent executes the enabled action with the highest reward which can be `SLOWER`, for example. The environment updates accordingly, and the reward is returned to the RL agent. Without a safety shield, the RL agent could predict `FASTER` with the highest reward and execute the action although it could be evaluated as unsafe.

[2] Note that at least one operation must always be enabled as discussed before.

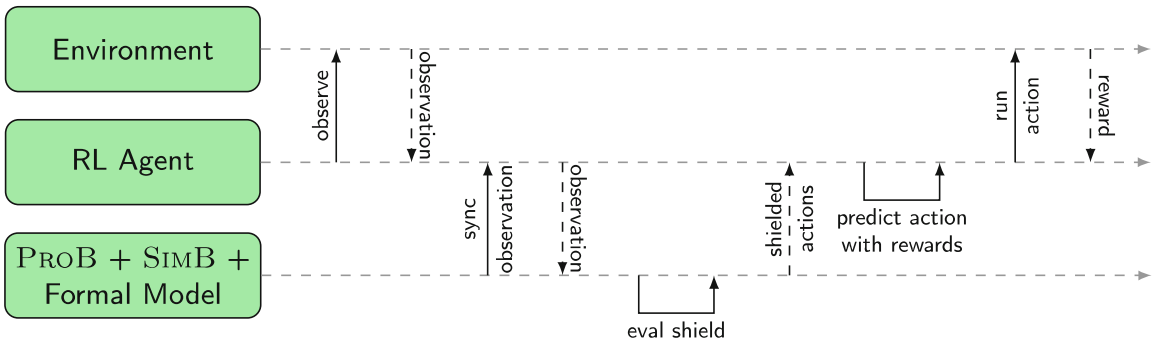


Fig. 3. Shielding the RL Agent with a Formal Model. The general control loop captures the current environmental state over which a set of enabled actions are computed by ProB, matching the safety shield's specification. The RL agent chooses the enabled action which has the highest reward for execution.

Section 4 shows that manually encoded safety shields can improve the RL performance over unshielded agents. However, we do not promote manually encoded shields but rather demonstrate the possibility of using a formal specification as a shield. In cases where this is not suitable, we recommend the synthesis of safety shields based on safety guarantees instead [12].

3.3 Validatability and Verifiability

This work facilitates simulating and reasoning about the RL agents' execution runs, despite their black-box nature. Based on a single execution run, one can evaluate the behavior with trace replay. If the agent behaves correctly in a critical situation, we can understand which decisions were particularly important. We can also assess errors leading to a safety-critical situation. With trace replay, one can evaluate which dangerous scenarios are avoided by safety shields. If the execution of an operation in a trace is blocked by a safety shield, the safety shield was effective in avoiding this particular dangerous scenario.

Given multiple execution runs, one can apply statistical validation techniques, e.g., estimation of certain values (probabilities, averages, and sums) and the likelihood of certain properties. This allows us to validate the choice of the reward function as well as the behavior and impact of safety shields.

By encoding expected domains for the variables' values after executing an operation, this work allows us to validate the implementation of the RL agent. We can also validate that the RL agent and its environment match the encoded domains. Consequently, the formal model together with the encoded safety shield can be seen as an over-approximation of the RL agent and its environment. In the future, we intend to use those validated domains as assumptions to (1) prove the formal model under these assumptions, and (2) also restrict the state space to make model checking easier to apply. With these techniques, one could then check safety properties (including invariants) on the formal model. When the formal model, and thus also the safety shield fulfill the safety property, one can

conclude that the safety property is enforced for the RL agent. As the formal model works as an over-approximation, this does not apply to liveness properties.

4 Case Study

We applied this work's methodology to various case studies which are available online[3]. In this section, we focus on using this technique to validate a highway environment RL agent [15]. First, we present the formal B model. We then describe how we train the agent, and how we apply SimB's simulation and statistical validation. We then apply trace replay, and domain-specific visualization to reason about the agent's decisions.

4.1 Formal B Model for Highway Environment

In the formal B model, we define variables storing the set of present vehicles (`PresentVehicles`), and total functions mapping each vehicle to its respective x and y-coordinates (`VehiclesX`, `VehiclesY`), and its velocities (`VehiclesVx`, `VehiclesVy`) and accelerations (`VehiclesAx`, `VehiclesAy`) in x and y-directions. The accelerations are computed from the current and previous observations wrt. the elapsed time between these two observations (one second). To make the formal model easier to understand, we define a set of `Vehicles` which includes the `EgoVehicle`. We further introduce a `Crash` and a `Reward` variable for validation purposes. Note that the encoding of the formal model in this section differs from the more abstract illustration described in Sect. 3.1.

Table 1. Encoding of Shield for Highway Agent

Action	Disabling Condition (Guard)
`LANE_LEFT`	Action is not executable if there is a vehicle on a lane further left which (1) is between 10 m and 30 m in front and drives slower (2) is between 10 m behind and 10 m in front (3) is between 10 m and 20 m behind and drives faster
`LANE_RIGHT`	Action is not executable if there is a vehicle on a lane further right which (1) is between 10 m and 30 m in front and drives slower (2) is between 10 m behind and 10 m in front (3) is between 10 m and 20 m behind and drives faster
`FASTER`	Action is not executable if distance to front vehicle is less than 40 m
`IDLE`	Action is not executable if distance to front vehicle is less than 30 m
`SLOWER`	Action is not executable if distance to front vehicle is less than 10 m and (1) `LANE_LEFT` is enabled or (2) `LANE_RIGHT` is enabled

[3] https://github.com/hhu-stups/reinforcement-learning-b-models.

Listing 2. `FASTER` Operation in B Model for Highway Environment; each vehicle's position corresponds to its center, each vehicle's length is 5 m, each vehicle's width is 2 m; therefore we encode $[0.0, 45.0]$ in x-direction and $[-3.5, 3.5]$ in y-direction to formulate that the distance to the vehicle in front is less than 40 m.

```
1  FASTER =
2  PRE
3    EgoVehicle ∈ dom(VehiclesVx) ∧
4    ¬(∃v. (v ∈ PresentVehicles \ {EgoVehicle} ∧
5    VehiclesX(v) > 0.0 ∧ VehiclesX(v) < 45.0  ∧
6    VehiclesY(v) < 3.5 ∧ VehiclesY(v) > -3.5))
7  THEN
8    Crash :∈ BOOL ||
9    PresentVehicles :| (PresentVehicles ∈ ℙ(Vehicles) ∧
10     EgoVehicle : PresentVehicles) ||
11   VehiclesX :∈ Vehicles → ℝ ||
12   VehiclesY :∈ Vehicles → ℝ ||
13   VehiclesVx :| (VehiclesVx ∈ Vehicles → ℝ ∧
14     (Crash = FALSE ⟹
15     VehiclesVx(EgoVehicle) ≥
16     VehiclesVx'(EgoVehicle) - 0.05)) ||
17   VehiclesVy :∈ Vehicles → ℝ ||
18   VehiclesAx :| (VehiclesAx ∈ Vehicles → ℝ ∧
19     (Crash = FALSE ⟹ VehiclesAx(EgoVehicle) ≥ -0.05)) ||
20   VehiclesAy :∈ Vehicles → ℝ ||
21   Reward :∈ ℝ
22 END
```

Corresponding to the agent's action space, we encoded 5 actions into the formal B model: `IDLE`, `LANE_LEFT`, `LANE_RIGHT`, `FASTER`, and `SLOWER`. Table 1 shows the description of the guards for all operations that we use as safety shield in our experiments. The `SLOWER` action is guaranteed to be enabled if no other guard would hold. Listing 2 shows the `FASTER` operation in our formal model with a safety shield. The guard (see lines 3–6) for shielding the `FASTER` action states that `FASTER` is not enabled if the distance to the vehicle in front is less than 40 m. As each vehicle's position corresponds to its center, and its length is 5 m, we encode 45 m in the formula. In lines 13–16, we encode that the expected speed remains the same or increases with a tolerance of –0.05 m/s. Likewise, the acceleration should be positive with the same tolerance (see lines 18–19).

4.2 Training the Agents

We compare two trained DQN agents for the environment, both are trained over the `highway-fast-v0` environment with three lanes. The first agent uses default configurations for the environment and the reward function. We will refer to this agent as Base agent. The reward function rewards the agent based on the resulting environment state caused by its last action. The environment's default

rewards are –1 for a collision, 0.1 when the agent is on the right-most lane, and 0.0–0.5 for a speed between 20–30 m/s (linearly scaled over the speed interval).

As it turned out, the agent's driving behavior proved to be rather risky, preferring speed over collision avoidance in certain cases and thus ending up with a high collision rate of almost 60 % (see Table 3). In response, we changed the penalty for collisions from –1 to –2. We also adjusted the reward for driving on the right-most lane from 0.1 to 0.2 to further the desired behavior of prioritizing the right-most lane. The agent trained with this altered reward function will be referred to as Higher Penalty.

For the DQN, we used a neural network with two hidden layers of 256 neurons each and a learning rate of 0.0005. The discount factor was set to 0.9 which affects the value of future rewards [31]: A reward received in k steps will only be 0.9^{k-1} times as valuable as if received immediately. The exploration rate decayed linearly from 1.0 to 0.05 within the first 6000 of a total of 20,000 training steps, indicating the ratio of actions which are taken randomly rather than following the thus far learned policy. This randomness is meant to overcome local maxima in the learned policy by regularly bypassing greedy behavior. The agents were each trained within 15 min.

4.3 Statistical Validation

Now, we apply SimB's statistical validation techniques to validate safety properties for the highway agent. For this, we evaluate 1000 execution runs per agent, once with and without a safety shield. We choose an episode length of 60 s for each run with a frequency of one observation per second. An episode might end earlier than 60 s if an accident occurs.

To estimate the RL agents' quality, we first gathered statistics over the resulting traces to get a feeling for how well the agents act in the first place. We measured averages of episode length, speed, distance traveled per episode, time on the right lane per episode, and reward. The results are shown in Table 2. One can see that Higher Penalty increases the average episode length to over 53 s, an increase of 14 s (+36.5 %) to Base. This indicates that the higher penalty was indeed a sensible choice. Further, we already see the benefits of shielding.

Table 2. Estimation of Average Values, Application of SimB Validation Techniques, and the Result of Validation; Values represent average metric values with standard deviation.

Metric	Base		Higher Penalty	
	no shield	with shield	no shield	with shield
Episode Length	38.85 ± 22.41	56.71 ± 11.47	53.02 ± 15.32	59.16 ± 5.54
Velocity [m/s]	23.37 ± 2.17	21.49 ± 0.94	21.14 ± 0.79	20.95 ± 0.63
Distance [m]	876.35 ± 477.62	1213.30 ± 244.48	1117.18 ± 321.71	1238.04 ± 122.12
On Right Lane [s]	31.69 ± 22.29	42.26 ± 20.51	47.07 ± 17.85	48.73 ± 17.52
Total Reward	30.41 ± 17.39	42.88 ± 8.86	39.90 ± 11.77	44.20 ± 4.42

Table 3. Safety Properties, Application of SimB Validation Techniques, and the Result of Validation. Percentages represent ratio of measured traces fulfilling the safety property.

Safety Property		Base		Higher Penalty	
		no shield	with shield	no shield	with shield
SAF1:	The agent must avoid collisions with other vehicles	45.4 %	91.8 %	78.5 %	97.4 %
SAF2:	The agent must drive faster than 20 m/s	93.4 %	91.4 %	76.9 %	83.0 %
SAF3:	The agent must drive slower than 30 m/s	95.2 %	98.8 %	100.0 %	100.0 %
SAF4:	The agent should decelerate at a maximum of 5 m/s^2	100.0 %	100.0 %	100.0 %	100.0 %
SAF5:	The agent should accelerate at a maximum of 5 m/s^2	100.0 %	100.0 %	100.0 %	100.0 %
SAF6:	To each other vehicle, the agent should keep a lateral safety distance of at least 2 m and a longitudinal safety distance of at least 10 m	6.4 %	49.2 %	41.6 %	70.5 %

Table 3 lists the safety properties we validated with SimB and the corresponding results for both agents with and without safety shields. The safety properties **SAF1**–**SAF6** cover the following aspects:

- **SAF1** is the main property and states that the agent must avoid collisions with other vehicles.
- **SAF2** and **SAF3** check that the agent drives with an appropriate speed.
- **SAF4** and **SAF5** check that the agent does not change speed by acceleration or braking abruptly.
- **SAF6** check that the agent should maintain appropriate distances from other cars to have enough room for reactions when accelerating, braking, and switching lanes.

Note that the validation objectives are not necessarily favored by the reward function, i.e., the agent is unaware of these specifications. For instance, **SAF6** is not rewarded during training. We intentionally validate untrained properties to show how the approach might capture such instances.

When evaluating **SAF1**, we found that the RL agent causes significantly fewer accidents if it is penalized more severely for accidents during training. We are also able to reduce the accident rate by encoding a safety shield. Especially for Base, the accident rate with a safety shield could be reduced to be safer than Higher Penalty without a safety shield. Despite the safety shield, collisions still occur in Higher Penalty. From the corresponding simulated traces, our

technique discovered that almost all scenarios with collisions consist of the ego vehicle approaching another vehicle in front while performing `SLOWER` and driving at the set minimum speed of 20 m/s: the front vehicle drives even slower leading to the collision. Our technique also discovered that setting a lower minimum speed, e.g. 19 m/s, is also not an appropriate solution. In this case, the ego vehicle sometimes drives slower than all other vehicles which leads to all other vehicles driving away at the front of the highway. This means the ego vehicle drives alone at the back of the highway without any collisions. There is also another rare scenario leading to a collision (in the experiments it was 1 of the 1000 simulated traces): the ego vehicle collides with another vehicle on the other side of the highway, i.e. the ego vehicle and another vehicle drive in the opposite outer lanes and both switch to the center lane simultaneously.

Driving too slow does not seem to be a factor in crashes (see **SAF2**). Sometimes, the actual speed might be slightly below the desired minimum speed, especially for Higher Penalty. While this seems to work against the environment's specification, we did not correct this with the safety shield, as there might be situations where it is sensible to brake and drive slower. Furthermore, Base agent sometimes drives slightly too fast, i.e., exceeding the speed limit of 30 m/s (see **SAF3**). As shown in the values for **SAF6**, it seems that maintaining safe distances is significantly more important to avoid accidents rather than exceeding the speed limit. In all four variations, the RL agent never accelerates or decelerates heavily, i.e., **SAF4** and **SAF5** are never violated. This is to be expected as the encoded acceleration range for the agent is [–5,5] by default. Thus, with the validation of **SAF4** and **SAF5**, we also validated the implementation of the RL agent. Relating the validation results to Table 2 again, we see that with safety shields:

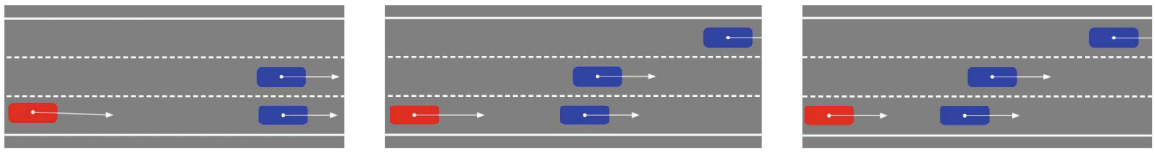
- The average speed is slower, but the distance traveled and the average episode length are greater. An interesting result here is that Base with a safety shield achieves a lower crash rate than Higher Penalty without a safety shield even with a higher average speed.
- The safety distances are maintained more often which seems to be the main reason for fewer crashes. Especially, Base with a safety shield maintains safety distances better than Higher Penalty without a safety shield.
- The cumulative reward is higher with a smaller standard deviation.
- The agent drives on the right lane more often.

Thus, our results highlight the safety capabilities of the employed shield and how pre-shielding can alleviate shortcomings during training. This helped us to calibrate the reward function better. Note that this work does not demonstrate that the manual encoding of the safety shield is perfect; in the future, we will consider shield synthesis [12]. However, we show that one can use a formal specification as a safety shield and that it achieves better RL performance than without.

4.4 Validation by Trace Replay

Now, we discuss validating the agent's behavior with trace replay, highlighting the role of safety shields. For easier understanding, we employ a domain-specific visualization [41] for the highway environment. We focus on two different, observed scenarios[4].

Figure 4 shows a scenario where the ego vehicle approaches another vehicle and slows down. Here, the agent was able to detect the vehicle in front and brake in time. The safety distance to the vehicle in front is hence kept and an accident could be avoided. Further, the RL agent seems to be aware of another vehicle in the center lane as it decides to slow down rather than switch lane. The scenario shown in Fig. 4 was simulated without safety shields. When re-playing this trace with safety shields being activated, the trace is still feasible. So, in this scenario, the RL agent behaves correctly without intervention by safety shields.



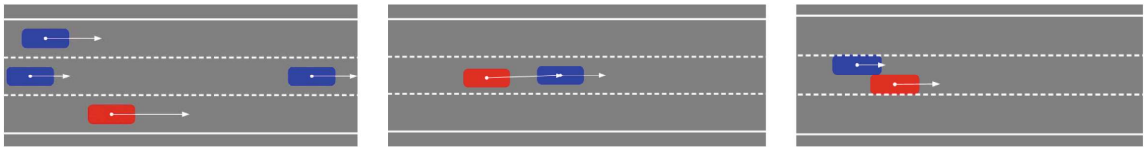
(a) Ego Vehicle Approaches (b) Ego Vehicle Slows Down (c) Ego Vehicle Slows Down

Fig. 4. Example for Approaching Scenario; white arrows show the direction of the velocity vector.

A second scenario is shown in Fig. 5. Here, the agent switches to the center lane while keeping a high velocity. After switching, the ego vehicle has to slow down as it is approaching another vehicle in front. As the agent does not brake in time, it collides with the vehicle in front. This scenario was also simulated without safety shields. When trying to re-play the trace with safety shields, the trace is not feasible anymore, especially when the RL agent tries to execute `LANE_LEFT`. Thus, a collision could have been avoided in this scenario by using safety shields.

Of course, the question of why the agent behaves in the observed manner cannot be answered completely. While the RL agent seems to behave in certain ways that correspond to similar human intuition, there still is no way to properly find reason in the agent's behaviors. This is due to the black box nature of neural networks underlying our DQN approach. While explainable AI methods from research [35] might offer insights, there are no guarantees they may accurately capture black box agents [26] and different explainers might even yield conflicting explanations. These problems with explainable AI emphasize the need for proper validation tools for RL agents, as outlined in this work.

[4] A scenario is a sequence of events which alters the system's state. Scenarios as static exports [36] available at: https://hhu-stups.github.io/highway-env-b-model/.



(a) Ego Vehicle Drives on Right Lane

(b) Ego Vehicle Switches to Center Lane

(c) Ego Vehicle Crashes

Fig. 5. Example for Crash Scenario; white arrows show the direction of the velocity vector.

5 Related Work

This section compares this work with other works in the field of formal methods for AI, with a stronger focus on RL.

Justified Speculative Control (JSC) [6] is a technique to achieve safe RL with formal methods. In JSC, formal verification results are obtained and integrated into the RL agent's controller. The verification results also provide a set of safe actions from which an RL agent can choose for execution. In our work, the RL agent also chooses from a set of safe operations which are computed from the manually encoded operations' guards in the formal model. While the formal model in our work could be verified for safety properties (depending on the model's state space), the creation of the shield is driven by requirements rather than verification results. Still, we can detect and avoid dangerous situations.

Sha [29] presented an approach to "use simplicity to control complexity" in which a simpler system monitors a complex system at runtime, and intervenes when certain rules are violated. Sha's concept is independent of reinforcement learning; the given example is about a complex Boeing flight system that was checked for laws by a simple, reliable controller. Based on Sha's concept, Phan et al. [22] presented a *neural simplex architecture* (NSA) for reinforcement learning. The NSA consists of a pre-certified *decision module* which switches between the complex unverified *neural controller* and a verified *baseline controller* if the former tries to execute a potentially dangerous action. Referring to Sha, the RL agent can be viewed as the complex system, and the formal model with the safety shield as the simple controlling system in our work. Furthermore, the safety shield influences every decision for the RL agent as ProB uses the formal model to compute actions that are deemed to be safe.

Shield synthesis [12] is a runtime verification technique which also aims to achieve safe RL. After modeling the environment as a Markov Chain, a shield is synthesized which may take over the agent's decision-making for a (possibly limited) number of steps. The shield acts once the probability of reaching an unsafe state shortly exceeds a given threshold. In our technique, we encoded shields by hand rather than synthesizing them. While the burden of precisely formulating the shielding conditions is now placed on the modeler, we can assume the RL agent's internal decision-making process as black box. However, we do not guarantee that the shield will be returning control to the agent eventually. In shield synthesis, there is also the concept of enforcing temporal properties,

especially LTL properties [4]. Assuming that the formal model's state space is finite, one could also use ProB's LTL model checker [23] in our approach, to verify LTL properties on the RL agent (with shielding). When the formal model fulfills a safety property, then we know that this safety property is also enforced for the RL agent. However, this is not the case for liveness properties.

Deep RL is implemented using neural networks for which there are also verification approaches [8,9,28], including techniques such as abstract interpretation [7], SMT solving [10], and proving [25]. Our work mainly focuses on validation and does not yet tackle the challenge of verifying the RL agent extensively. In the future, we should investigate how to achieve and guarantee better safety of RL agents in our approach.

Search-based testing [33] is a technique which uses a depth-first search to find safety-critical states. The RL agent is then brought into a situation close to the safety-critical state to test how well it avoids this state. The technique also applies *fuzz testing* to achieve better coverage of the RL agent's behavior. *Differential safety testing* [32] is another technique to test RL agents for safety, which makes use of automata learning [18,19], probabilistic model checking [3], and statistical methods. With Monte Carlo simulation, our work simulates multiple different scenarios. Based on the resulting execution runs, our work can estimate certain values and compute the likelihood of fulfilling certain safety properties. The results are then used to evaluate and improve the safety shield and the reward function. However, we have not yet navigated the RL agent into critical situations for testing purposes.

Wang et al. [39] presented a safety-falsification method which works as an adversary for the RL agent. The technique uses metric temporal logic formulas to enforce the RL agent to violate safety properties. As these properties are difficult to integrate into the reward function, safety-falsification helps the RL agent to train adversarial behavior. As we do not use a safety shield during training, our RL agents also experience the consequences of bad behaviour in the form of reward penalties.

Shalev-Shwartz et al. [30] presented a formal model for safe behavior of self-driving cars, called responsibility-sensitive safety (RSS). This model was later extended and translated to Event-B [11]. The rules of RSS in general and the Event-B model, in particular, could be integrated as safety shields into our approach in the future.

6 Conclusion and Future Work

This work presented a technique to validate RL agents with formal methods tools and techniques. We create a formal model at a high-level abstraction and link it with the RL agent. This allows to use the formal specification as a safety shield for the RL agent. Furthermore, the formal model encodes the RL agent's expected external behavior, i.e., the RL agent's actions and its environment. It is then possible to apply validation techniques like trace replay, simulation, or statistical validation.

In this work, we successfully demonstrated our technique using the formal B method with the tools ProB and SimB on a highway environment. With trace replay, and real-time simulation, we can replay the agent's situation and reason about its decisions. Here, we also demonstrated that dangerous scenarios are avoided by safety shields in the formal model. Applying statistical validation techniques, we can estimate the likelihood of fulfilling various safety requirements, e.g., the likelihood of crashes or not maintaining safety distances. We also estimate certain values, e.g., the average reward, the average speed, the average distance of one episode, the average time on the right lane of one episode, or the average episode length of the RL agent on the highway. With the gained knowledge, we improved safety shields which again increased safety. Safety shields were effective in reducing the likelihood of crashes at the cost of reducing the average velocity, overall increasing the safety of the model. With the manually encoded safety shields in the formal model, we also achieve higher rewards for the agent. We were even able to validate the reward function, highlighting where we needed to adjust the respective weights. Furthermore, we were able to validate the implementation of the RL agent. All models, including highway environment, are available online at:

https://github.com/hhu-stups/reinforcement-learning-b-models

While our approach enables various validation techniques, verification has yet not been tackled actively. We aim to validate and better understand the RL agent's behavior to collect assumptions about the agent and its environment. Based on this, we plan to verify the model with techniques like model checking or proving. Assuming that safety properties are fulfilled for the formal model, we can also conclude these properties for the RL agent, as the formal model is encoded as an over-approximation of the RL agent. As future work, one could further investigate how our approach can be extended by shielding over LTL properties.

Acknowledgements. We would like to thank Davin Holten for his initial experiments showing the feasibility of ProB's techniques — especially of SimB — for the highway RL agent.

References

1. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge (2010). https://doi.org/10.1017/CBO9781139195881
2. Abrial, J.R., Hoare, A.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, Cambridge (2005). https://doi.org/10.1017/CBO9780511624162
3. Aichernig, B.K., Tappler, M.: Probabilistic black-box reachability checking (extended version). Form. Methods Syst. Des. **54**(3), 416–448 (2019). https://doi.org/10.1007/s10703-019-00333-0
4. Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., Topcu, U.: Safe reinforcement learning via shielding. In: Proceedings AAAI, pp. 2669–2678. AAAI Press (2018). https://doi.org/10.1609/aaai.v32i1.11797

5. Bendisposto, J., et al.: ProB2-UI: a java-based user interface for ProB. In: Lluch Lafuente, A., Mavridou, A. (eds.) FMICS 2021. LNCS, vol. 12863, pp. 193–201. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85248-1_12
6. Fulton, N., Platzer, A.: Safe reinforcement learning via formal methods: toward safe control through proof and learning. In: Proceedings AAAI, pp. 6485–6492. AAAI Press (2018). https://doi.org/10.1609/aaai.v32i1.12107
7. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: Ai2: safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE Symposium on Security and Privacy (SP), pp. 3–18. IEEE (2018). https://doi.org/10.1109/SP.2018.00058
8. Huang, X., Kwiatkowska, M., Wang, S., Wu, M.: Safety verification of deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 3–29. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_1
9. Huang, X., Ruan, W., Tang, Q., Zhao, X.: Bridging formal methods and machine learning with global optimisation. In: Riesco, A., Zhang, M. (eds.) Formal Methods and Software Engineering. ICFEM 2022. LNCS, vol. 13478, pp. 1–19. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-17244-1_1
10. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 97–117. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_5
11. Kobayashi, T., Bondu, M., Ishikawa, F.: Formal modelling of safety architecture for responsibility-aware autonomous vehicle via event-b refinement. In: Proceedings FM'2023, pp. 533–549 (2023). https://doi.org/10.1007/978-3-031-27481-7_30
12. Könighofer, B., Lorber, F., Jansen, N., Bloem, R.: Shield synthesis for reinforcement learning. In: Margaria, T., Steffen, B. (eds.) ISoLA 2020. LNCS, vol. 12476, pp. 290–306. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-61362-4_16
13. Krings, S.: Towards Infinite-State Symbolic Model Checking for B and Event-B. Ph.D. thesis, Heinrich Heine Universität Düsseldorf, August 2017
14. Landers, M., Doryab, A.: Deep reinforcement learning verification: a survey. ACM Comput. Surv. **55**(14s) (2023). https://doi.org/10.1145/3596444
15. Leurent, E.: An Environment for Autonomous Driving Decision-Making (2018). https://github.com/eleurent/highway-env
16. Leuschel, M., Butler, M.: ProB: a model checker for B. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003. LNCS, vol. 2805, pp. 855–874. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45236-2_46
17. Leuschel, M., Butler, M.: ProB: an automated analysis toolset for the B method. STTT **10**(2), 185–203 (2008). https://doi.org/10.1007/s10009-007-0063-9
18. Mao, H., Chen, Y., Jaeger, M., Nielsen, T.D., Larsen, K.G., Nielsen, B.: Learning Markov decision processes for model checking. In: Proceedings QFM, pp. 49–63. EPTCS 103, Open Publishing Association (2012). http://dx.doi.org/10.4204/EPTCS.103.6
19. Mao, H., Chen, Y., Jaeger, M., Nielsen, T.D., Larsen, K.G., Nielsen, B.: Learning deterministic probabilistic automata from a model checking perspective. Mach. Learn. **105**(2), 255–299 (2016). https://doi.org/10.1007/s10994-016-5565-9
20. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015). https://doi.org/10.1038/nature14236
21. Peer, E., Menkovski, V., Zhang, Y., Lee, W.J.: Shunting trains with deep reinforcement learning. In: Proceedings SMC, pp. 3063–3068. IEEE (2018). https://doi.org/10.1109/SMC.2018.00520

22. Phan, D.T., Grosu, R., Jansen, N., Paoletti, N., Smolka, S.A., Stoller, S.D.: Neural simplex architecture. In: Lee, R., Jha, S., Mavridou, A., Giannakopoulou, D. (eds.) NFM 2020. LNCS, vol. 12229, pp. 97–114. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-55754-6_6
23. Plagge, D., Leuschel, M.: Seven at one stroke: LTL model checking for high-level specifications in B, Z, CSP, and more. STTT **12**(1), 9–21 (2010). https://doi.org/10.1007/s10009-009-0132-3
24. Razzaghi, P., et al.: A Survey on Reinforcement Learning in Aviation Applications (2022). https://doi.org/10.48550/arXiv.2211.02147
25. Ruan, W., Huang, X., Kwiatkowska, M.: Reachability analysis of deep neural networks with provable guarantees. In: Proceedings IJCAI, pp. 2651–2659 (2018). https://doi.org/10.24963/ijcai.2018/368
26. Rudin, C.: Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nat. Mach. Intell. **1**(5), 206–215 (2019). https://doi.org/10.1038/s42256-019-0048-x
27. Sallab, A., Abdou, M., Perot, E., Yogamani, S.: Deep reinforcement learning framework for autonomous driving. Electron. Imaging **29**(19), 70–76 (2017). https://doi.org/10.2352/ISSN.2470-1173.2017.19.AVM-023
28. Seshia, S.A., Sadigh, D., Sastry, S.S.: Toward verified artificial intelligence. Commun. ACM **65**(7), 46–55 (2022). https://doi.org/10.1145/3503914
29. Sha, L.: Using simplicity to control complexity. IEEE Softw. **18**(4), 20–28 (2001). https://doi.org/10.1109/MS.2001.936213
30. Shalev-Shwartz, S., Shammah, S., Shashua, A.: On a formal model of safe and scalable self-driving cars. CoRR **abs/1708.06374** (2017). http://arxiv.org/abs/1708.06374
31. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (2018)
32. Tappler, M., Aichernig, B.K.: Differential safety testing of deep RL agents enabled by automata learning. In: Steffen, B. (eds.) Bridging the Gap Between AI and Reality. AISoLA 2023. LNCS, vol. 14380, pp. 138–159 Springer, Cham (2024). https://doi.org/10.1007/978-3-031-46002-9_8
33. Tappler, M., Cano Cordoba, F., Aichernig, B., Könighofer, B.: Search-based testing of reinforcement learning. In: Proceedings IJCAI, pp. 503–510 (2022). https://doi.org/10.24963/ijcai.2022/72
34. Tran, H.D., Cai, F., Diego, M.L., Musau, P., Johnson, T.T., Koutsoukos, X.: Safety verification of cyber-physical systems with reinforcement learning control. ACM Trans. Embed. Comput. Syst. **18**(5s), 1–22 (2019). https://doi.org/10.1145/3358230
35. Vouros, G.A.: Explainable deep reinforcement learning: state of the art and challenges. ACM Comput. Surv. **55**(5), 1–39 (2022). https://doi.org/10.1145/3527448
36. Vu, F., Happe, C., Leuschel, M.: Generating interactive documents for domain-specific validation of formal models. STTT (2024). https://doi.org/10.1007/s10009-024-00739-0
37. Vu, F., Leuschel, M.: Validation of formal models by interactive simulation. In: Glässer, U., Creissac Campos, J., Méry, D., Palanque, P. (eds.) ABZ 2023. LNCS, vol. 14010, pp. 59–69. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-33163-3_5
38. Vu, F., Leuschel, M., Mashkoor, A.: Validation of formal models by timed probabilistic simulation. In: Raschke, A., Méry, D. (eds.) ABZ 2021. LNCS, vol. 12709, pp. 81–96. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77543-8_6

39. Wang, X., Nair, S., Althoff, M.: Falsification-based robust adversarial reinforcement learning. In: Proceedings ICMLA, pp. 205–212. IEEE (2020). https://doi.org/10.1109/ICMLA51294.2020.00042
40. Watkins, C.J., Dayan, P.: Q-learning. Mach. Learn. **8**, 279–292 (1992). https://doi.org/10.1007/BF00992698
41. Werth, M., Leuschel, M.: VisB: a lightweight tool to visualize formal models with SVG graphics. In: Raschke, A., Méry, D., Houdek, F. (eds.) ABZ 2020. LNCS, vol. 12071, pp. 260–265. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-48077-6_21
42. Zhou, Z.-H.: Machine Learning. Springer, Singapore (2021). https://doi.org/10.1007/978-981-15-1967-3