Case Study Report: Verification Guided RL

Riccardo De Sanctis

November 8, 2025

Abstract

This report describes Riccardo De Sanctis' personal case study for Sapienza Computer Science FMAISE course, where SPIN model checker is used iteratively during training to remove unsafe state-action transitions in the FrozenLake environment. The goal is to reduce unsafe exploration, improve sample efficiency, and provide formal guarantees about safety of the resulting action space provided to an RL agent. The pipeline works similarly to a controller.

1 Problem statement

Reinforcement Learning (RL) agents learn from scalar rewards supplied by the environment. When rewards are sparse or incomplete, agents may exhibit unsafe or undesired behaviors during training such as unsafe exploration (e.g. falling into holes), reward hacking (i.e. achieving high scores in a way that is misaligned with the human designer's original intent or true goal) and poor exploration stemming from sparse rewards and leading to long training time and sample inefficiency. The objective of this work is to replace or augment reward-driven feedback with an iterative model-checking step that prunes actions that can lead to unsafe states, thus creating a reduced, safe action space for subsequent RL training.

Intuitively, pruning avoids letting the agent perform transitions that have been formally proven to lead directly to unsafe states.

Figure 1: Default 4×4 FrozenLake Environment

2 System model

2.1 Environment

We model the standard FrozenLake environment (OpenAI Gymnasium): an $n \times n$ grid with cells of types Start (S), Frozen (F), Hole (H), and Goal (G). The agent has four nominal actions in each non-terminal cell (S and F cells) and Goal cell: Left (L), Down (D), Right (R), Up (U). The environment can be deterministic (non-slippery) or stochastic (slippery). In slippery

Figure 2: Slippery FrozenLake Environment Action Outcome Illustration

environments, the agent will move in intended direction with probability of 1/3 else will move in either perpendicular direction with equal probability of 1/3 in both directions.

2.2 SPIN verification pipeline

SPIN interfaces with Gymnasium through a Python proxy. The environment map is extracted from Gymnasium using the description (`desc`) argument of the FrozenLake environment; a Promela file is generated representing the respective environment. Note that initially the set of safe actions of each cell includes all the four actions (L,D,R,U):

- For each cell that is not a Hole, all safe actions are represented non-deterministically as branching conditions in the Promela file:

```
::  (x==0 && y==0) ->
  if
    ::  (x + 1, y + 1) == OOB -> Record Violation /* Move Right into Wall */
    ::  (x + 1, y + 1) == Hole -> Record Violation /* Move Right into Hole */
    ::  (x + 1, y + 1) == Safe -> x += 1; y += 1 /* Move Right is Safe*/
    ::  ...  /* Same for other action directions that are in safe_actions */
```

- Out-of-bound (OOB) conditions are handled differently depending on whether the environment is slippery: in slippery envs they are allowed.

Iterations of the pruning process consists of:

- Generating the Promela file (as described above, additional implementation details are given in Section 4): `generate_promela(desc, safe_actions, max_steps)`.
- Invoking SPIN for formal verification on the Promela file: `spin -a promela_file` to generate `pan.c` then `spin -v promela_file` to perform SPIN verbose trace verification.
- Invalid actions are recorder in the SPIN trace. `OFF: x=0;y=0;Left`.
- Parse the SPIN trace in python to remove unsafe actions from the valid actions set: `safe_actions[0,0].drop("Left")`
- Next Promela file generated will use the updated set of valid actions. `safe_actions[0,0] = ["Down", "Right", "Up"]`
- The process is repeated iteratively until no further unsafe actions are detected.

3 Safety properties

We consider reachability and safety LTL-style properties mapped to SPIN for verification. Properties used in the case study are:

- **Reachability from start to goal:** `F` *goal* (eventually reachinig goal from starting state).
- **Safety (avoid holes):** `G` $\neg$ *hole* (globally avoid hole).
- **Safety (avoid walls)**: `G` $\neg$ *OOB* (globally avoid walls). Enabled only in non-slips envs.

The iterative refinement enforces that any action leading to a violation is eliminated from the global `safe_actions` mapping. SPIN detects if the task can be solved (Goal reacheable) performing only safe actions.

4 Implementation notes

The provided Python implementation (Gymnasium + SPIN invocation) follows these steps:

1. Generate a parametrised Promela `proctype Agent(int sx; int sy)` that starts in cell `sx,sy` and nondeterministically selects among `safe_actions` for each non terminal cell $\cup$ goal cell. Each branch records the attempted action and prints a short trace of unsafe outcomes and of agent positions. Different agents are spawned within the map to account for isolated group of cells, unreachable from the starting position (note that the starting cell is an environment parameter and may be customized).
2. Run `spin -a` to produce `pan.c`, compile it and run the verifier (`./pan`).
3. Parse the trace output: for each printed violation in the SPIN trace, remove the corresponding action in the global `safe_actions` mapping: note that for time efficiency reason, assertion that would block spin at first error are skipped and trace logs are used instead. In this way multiple actions can be removed within the same spin iteration, essential for efficiency of large map in particular. A next iteration unsafe actions that were not reached previously are now easier to identify.
4. Repeat until no new unsafe transitions are discovered. Perform a final iteration with only an agent at the start cell to ensure a reliable check for reachability to the goal. Note that multiple agents are instantiated and their interleaving is concurrent.

Some pragmatic implementation choices and assumptions:

- Holes are immediately reported on the SPIN trace; the generator avoids spawning agents at hole cells.
- For deterministic (non-slippery) dynamics, we treat out-of-bounds moves as unsafe.
- Slippery (stochastic) dynamics is modelled by adding perpendicular outcomes to each nominal action. Thus, resulting in additional branching options ($\times 3$) in the agent dynamics.
- The method is conservative: any action that can (possibly via nondeterministic outcomes) lead to a hole is removed. Actions leading to dead ends are kept to avoid over-pruning the action space and make the goal unreachable.

5 Experimental observations

In the FrozenLake case study the pruning step yields the following practical benefits (observed empirically):

- Significant reduction of available actions per state: from up to 4 nominal actions down to an often much smaller set (ranging from $\sim 30\%$ to $\sim 60\%$ of initial actions).

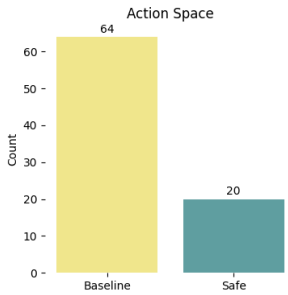

Figure 3: Slippery 4×4 FrozenLake Environment Safe Actions Reduction

- No episodes where the agent falls into holes during subsequent RL training.

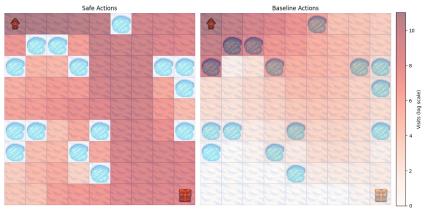

Figure 4: Slippery 9×9 FrozenLake Environment Heatmap

- Faster convergence of RL learners operating only over the pruned action set due to fewer wasted samples.

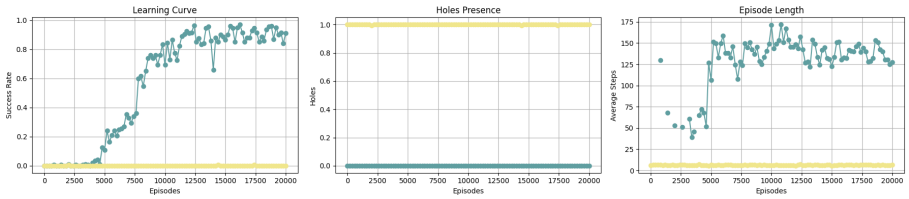

Figure 5: Slippery 9×9 FrozenLake Environment Learning Curves

- More stable policies across training runs given to smaller actions set. Baseline agent does not consistently reaches good behaviours across different runs, whereas safe agent is able to consistently replicate good runs even on larger maps.

6 Conclusions and Future Work

The iterative SPIN-guided pruning is a lightweight, formal-first way to enforce safety constraints before RL training. Next steps include:

- Extend the work to other RL environments.
- Investigating hybrid approaches that reintroduce pruned actions under monitored conditions (runtime shields) to avoid dead-ends or try to overcome them if safety can be traded-off.