Proceedings of the Thirty-Sixth International Conference on Automated Planning and Scheduling (ICAPS 2026)

Algorithms for Deciding the Safety of States in Fully Observable Non-deterministic Problems

Johannes Schmalz, Chaahat Jain

Saarland University, Germany

Abstract

Learned action policies are increasingly popular in sequential decision-making, but suffer from a lack of safety guarantees. Recent work introduced a pipeline for testing the safety of such policies under initial-state and action-outcome non-determinism. At the pipeline's core, is the problem of deciding whether a state is safe (a safe policy exists from the state) and finding *faults*, which are state-action pairs that transition from a safe state to an unsafe one. Their most effective algorithm for deciding safety, TarjanSafe, is effective on their benchmarks, but we show that it has exponential worst-case runtime with respect to the state space. A linear-time alternative exists, but it is slower in practice. We close this gap with a new *policy-iteration* algorithm iPI, that combines the best of both: it matches TarjanSafe's best-case runtime while guaranteeing a polynomial worst-case. Experiments confirm our theory and show that in problems amenable to TarjanSafe iPI has similar performance, whereas in ill-suited problems iPI scales exponentially better.

Technical Report — https://arxiv.org/abs/2603.15282

1 Introduction

Learned action policies are gaining traction in AI and AI planning, e.g., Mnih et al. (2015); Silver et al. (2016, 2018); Toyer et al. (2020). However, such policies come without any safety guarantees, creating demand for safety assurance methods. We focus on the safety-testing framework of Jain et al. (2025) (henceforth JEA) in fully observable non-deterministic (FOND) planning. In JEA's setting, a policy π is **safe** in a given state s if its execution will never lead to a **failure condition** ϕ_F, a state is **safe** if it has a safe policy, and **faults** are state-action pairs $(s, \pi(s))$ where s is safe but applying $\pi(s)$ non-deterministically leads to an unsafe state s'. Their framework takes a learned policy, finds paths therein that lead to fail states, and then finds faults by deciding the safety of states along the paths. These faults can be used to diagnose and repair the policy. For the core task of deciding state safety, JEA introduced TarjanSafe and demonstrated that it is very effective.

In this paper, we show that TarjanSafe has an *exponential worst-case time complexity w.r.t. the state space* using a pathological family of tasks where TarjanSafe expands the same states many times. Despite this limitation, TarjanSafe performs well in practice because it often explores only a small fraction of the state space. To contrast, we present Prop-$\mathcal{U}$, which follows techniques from reachability games (Diekert and Kufleitner 2022) and works by propagating unsafe states. Its worst case is linear, but it is unusable in practice because there are too many unsafe states to propagate. To address this gap we introduce iPI, a new algorithm based on **Policy Iteration (PI)** that combines a *polynomial worst-case runtime* while enjoying the practical speed of TarjanSafe; iPI starts with an initial policy π_{init} and attempts to prove its safety, making minimal changes to the policy as unsafe states are discovered; this makes iPI very efficient at finding safe policies similar to π_{init}, which occurs frequently. Finally, we confirm our theory experimentally. We use JEA's benchmarks, which mostly correspond to TarjanSafe's best case; there, iPI shows similar performance to TarjanSafe. Then, we introduce the *Flappy Bird* domain to showcase a problem ill-suited to TarjanSafe, and indeed iPI scales exponentially better. This places iPI as the current best algorithm for deciding state safety and gives valuable insight into what makes a good algorithm for this problem.

Our contributions are: a simplified formalism for FOND safety, new algorithms for deciding safety (Prop-$\mathcal{U}$ and iPI), a complexity analysis of TarjanSafe and our algorithms, and an empirical evaluation that shows iPI's effectiveness.

2 Background

We consider fully-observable non-deterministic (FOND) planning tasks (Daniele, Traverso, and Vardi 2000; Cimatti et al. 2003). We follow the formalism of JEA, but break it down into a simpler, equivalent formulation. A task's transition system is defined by $\Theta = \langle \mathsf{S}, \mathsf{A}, \mathcal{T}, \mathsf{S}_I \rangle$ where S is a finite set of states, A is a finite set of actions and we write $\mathsf{A}(s)$ to denote the set of actions applicable in state s, $\mathcal{T}$ is a non-deterministic transition function where $\mathcal{T}(s,a) \in \mathcal{P}(\mathsf{S}) \setminus \emptyset$ gives the set of states that may occur after applying a to s, $\mathsf{S}_I \subsetneq \mathsf{S}$ is a non-empty set of initial states.

Policies are partial functions $\pi : \mathsf{S} \to \mathsf{A}$ with the property that $\pi(s) \in \mathsf{A}(s)$ wherever defined. A policy π induces a **policy graph** $\Theta^\pi = \langle \mathsf{S}, \mathsf{A}, \mathcal{T}^\pi, \mathsf{S}_I \rangle$ with the same components as Θ except $\mathcal{T}^\pi$ has $\mathcal{T}^\pi(s,a) = \mathcal{T}(s,a)$ if $a = \pi(s)$ and is undefined otherwise. A **path** is a finite or infinite se-

quence of alternating states and actions $\sigma = s_0, a_0, s_1 \ldots$ that satisfies $a_i \in \mathsf{A}(s_i)$ and $s_{i+1} \in \mathcal{T}(s_i, a_i)$ for all $i \geq 0$ in σ. The set of all paths from a state s is *Paths*(s) and *Paths*$^\pi(s)$ is the set of paths from s in the policy graph Θ^π.

Typically, FOND tasks are also specified with a non-empty set of goal states $\mathsf{G} \subseteq \mathsf{S}$. We call such tasks $\Theta = \langle \mathsf{S}, \mathsf{A}, \mathcal{T}, \mathsf{S}_I, \mathsf{G} \rangle$ **goal-reaching FOND tasks**. To solve them we consider strong cyclic solutions, which we present as policies π s.t. following π from any initial state $s_I \in \mathsf{S}_I$ will eventually reach a goal in G assuming *fairness*; fairness ensures that each effect in $\mathcal{T}(s, a)$ is eventually triggered if a is applied to s often enough (Cimatti et al. 2003).

Recall that our setting is to identify whether a safe policy exists for states in a learned policy's envelope, so in this paper we focus on avoiding certain fail states rather than goal-reaching. Thus, we specify a non-empty set of **fail states** $\mathsf{S}_f \subseteq \mathsf{S}$, giving us **state-avoiding FOND tasks** $\Theta = \langle \mathsf{S}, \mathsf{A}, \mathcal{T}, \mathsf{S}_I, \mathsf{S}_f \rangle$.[1] A solution to such Θ is a policy π that never encounters any fail state, i.e., for all finite and infinite paths $\sigma \in \bigcup_{s_I \in \mathsf{S}_I} \mathit{Paths}^\pi(s_I)$, σ does not contain any states in S_f. State-avoiding FOND lets us express notions of safety, i.e., that a "bad" state should never happen, e.g., a self-driving car must never crash into a wall. A state s is called **unsafe** if no policy can guarantee that it avoids fail states from s, i.e., $\forall \pi \; \exists \sigma \in \mathit{Paths}^\pi(s) : \sigma \cap \mathsf{S}_f \neq \emptyset$. The set of unsafe states is called the unsafe region $\mathcal{U}$.

State-avoiding tasks can be transformed into goal-reaching ones, so state-of-the-art algorithms for state-reaching FOND can be applied to our setting, e.g., PR2 (Muise, McIlraith, and Beck 2024). However, we limit the scope of this paper to fundamental algorithms that are native to state-avoiding, letting us make a fairer comparison to JEA and perform sharper complexity analyses, thereby establishing a foundation for future work. Moreover, using their algorithms in our codebase is non-trivial because it requires a translation from JANI to variants of PDDL, which is an open line of research, e.g., (Klauck et al. 2018).

Now, we return to the problem formalisation. JEA look for safe policies in tasks that have goals, other terminal states, and fail states to avoid. Their policies must avoid the fail states, either by reaching a terminal state and stopping there, or by cycling in a way that avoids the fail states. JEA's tasks can be transformed into our state-avoiding tasks by adding a self-loop action to goal and terminal states, allowing policies to avoid fail states indefinitely by using these self loops rather than terminating. For the rest of this paper we focus on deciding the safety of individual states, so we only consider singleton initial states $\mathsf{S}_I = \{s_I\}$.

3 Algorithms

We start with TarjanSafe (JEA) and prove that it has a worst-case exponential runtime, but a good best-case that reflects its strong performance in practice. Then, we define Prop-$\mathcal{U}$, an algorithm that solves state-avoiding tasks in linear time, but has a worse best-case that makes it impractical. Finally, we introduce a PI algorithm and prove that it combines the best of both. We write $m = |\mathsf{A}|$, $n = |\mathsf{S}|$, $b = \max_{s \in \mathsf{S}, a \in \mathsf{A}(s)} |\mathcal{T}(s, a)|$ (branching factor of non-determinism), and $\pi_{\text{min-safe}}$ is a safe policy with a minimal number of states in its policy graph. We assume that $\mathsf{A}(s) \neq \emptyset$ for each state, so $m \geq n$. Our complexity results are summarised in tab. 1. Our best-case runtime analyses assume a safe policy exists; if no safe policy exists then the algorithms can be implemented with an early stop, in which case they terminate in one or two steps, i.e., $O(1)$.

[1] JEA use a failure condition ϕ_F, but for our discussion it is equivalent to consider a specified set of fail states $\mathsf{S}_f \subseteq \mathsf{S}$.

	Best case	Worst case
TarjanSafe	$\Theta(\lvert\pi_{\text{min-safe}}\rvert)$	$\Omega(2^m)$
Prop-$\mathcal{U}$	$\Theta(\lvert\mathsf{S}_f\rvert)$	$\Theta(bm)$
nPI, iPI	$\Theta(\lvert\pi_{\text{min-safe}}\rvert)$	$O(bmn)$

Table 1: Best-case (where a safe policy exists) and worst-case runtimes of the algorithms we consider. We use $m = |\mathsf{A}|$, $n = |\mathsf{S}|$, branching factor of non-determinism b, and $|\pi_{\text{min-safe}}|$ is a minimal safe policy's envelope size.

3.1 TarjanSafe

TarjanSafe is JEA's algorithm for finding FOND state-avoiding policies. It is a Depth First Search (DFS) with a labelling mechanism for states that have been proven safe or unsafe, and it detects Strongly Connected Components (SCCs) similarly to Tarjan's algorithm (Tarjan 1972). TarjanSafe's DFS is defined recursively: in its recursive cases, TarjanSafe iterates over $a \in \mathsf{A}(s)$ and calls itself on the successor states $\mathcal{T}(s, a)$, then it either marks s as unsafe if all these actions lead to unsafe states, or otherwise the algorithm assumes that s is safe and proceeds. Additionally, TarjanSafe detects SCCs and marks the SCC's root as safe if the SCC contains no unsafe states. Then, the recursion stops at s without making further recursive calls in the following cases: (1) s is a fail state ($s \in \mathsf{S}_f$); (2) s marked as unsafe; (3) s is a goal state (not relevant in our setting); (4) s marked as safe; (5) some action in $\mathsf{A}(s)$ leads only to states in the DFS `stack`. See the technical report for details. We motivate TarjanSafe's effectiveness with its best-case runtime and show that its worst case is exponential.

Theorem 1. *TarjanSafe has a best-case runtime of* $\Theta(|\pi_{\textit{min-safe}}|)$ *(if a safe policy exists).*

Proof sketch. In the best case, TarjanSafe will correctly guess $\pi_{\text{min-safe}}$ and traverse $\pi_{\text{min-safe}}$'s policy graph once with its DFS to prove that all states are safe.

Theorem 2. *TarjanSafe has a worst-case runtime of* $\Omega(2^m)$.

Proof. We present a class of problems where TarjanSafe requires on the order of 2^m steps, which proves that TarjanSafe's worst-case is asymptotically 2^m or worse, i.e., $\Omega(2^m)$. Consider the task in fig. 1 with 5 layers, and consider the generalised construction with d layers. This task has no fail, unsafe, nor goal states, so TarjanSafe's cases 1–3 will never trigger for any state. Case 4 can only be triggered at an SCC's root, which is only s_0, so this case only triggers when the algorithm terminates. Thus, the recursion can only bottom out at state s if s's applicable action leads to states

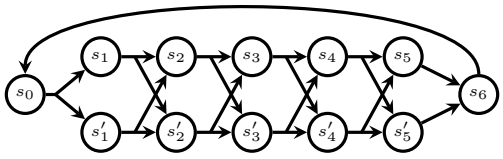

Figure 1: Non-deterministic task with 5 layers and 2^5 paths from s_0 to s_6. Similarly constructed tasks with d layers have 2^d paths from s_0 to s_{d+1}.

that are already on the `stack`, which in turn only happens at the state in the last layer, leading to s_0. This forces the algorithm to re-expand states and enumerate all paths. In our task with d layers, TarjanSafe must enumerate all paths from s_0 to s_{d+1} before it can terminate, which is 2^d paths. The problem has $2d+2$ states and $2d+2$ actions with a branching factor b of 2, so indeed the runtime of the algorithm is exponential w.r.t. the transition system size. □

3.2 Unsafety Propagation

Prop-$\mathcal{U}$ solves state-avoiding tasks in linear time by propagating the unsafe region. That is: (1) Prop-$\mathcal{U}$ starts with a subset of the unsafe region $\mathcal{Z} \leftarrow \mathsf{S}_f$, (2) it selects (s, a) such that $\mathcal{T}(s, a) \cap \mathcal{Z} \neq \emptyset$ and marks these (s, a) as unsafe, (3) if all actions in $\mathsf{A}(s)$ are unsafe then s gets added to $\mathcal{Z}$, (4) the algorithm iterates until no more (s, a) lead into $\mathcal{Z}$. Upon termination $\mathcal{Z} = \mathcal{U}$, and thereby, a safe policy exists from s iff $s \notin \mathcal{Z}$. Prop-$\mathcal{U}$ adapts the linear-time algorithm of Diekert and Kufleitner (2022) for solving reachability games.

Theorem 3. *Prop-$\mathcal{U}$ has a best-case runtime of $\Theta(|\mathsf{S}_f|)$ (if a safe policy exists) and a worst-case runtime of $\Theta(bm)$.*

Proof sketch. Prop-$\mathcal{U}$ adds each $s \in \mathcal{U}$ to $\mathcal{Z}$ precisely once. Prop-$\mathcal{U}$'s best case is $\mathcal{U} = \mathsf{S}_f$, giving $\Theta(|\mathsf{S}_f|)$. The worst case must consider that $s \in \mathcal{U}$ may be considered multiple times if there are $a \in \mathsf{A}(s)$ so that $\mathcal{T}(s, a) \cap \mathcal{U} \neq \emptyset$ in multiple steps. We take this into account with bm: the number of actions m and their non-deterministic branching b.

Prop-$\mathcal{U}$'s worst-case runtime of $\Theta(bm)$ is linear, since bm gives the number of deterministic transitions in our state space. This algorithm has the lowest worst-case complexity of the algorithms we consider in this paper, and it seems unlikely that it is possible to do better. However, in our setting, the unsafe region to propagate is very large (exponential w.r.t. problem description), making Prop-$\mathcal{U}$ impractical. More details about Prop-$\mathcal{U}$ are given in the technical report.

3.3 Policy Iteration (PI)

Our implementation of PI in alg. 1, named nPI, makes a nice tradeoff: it has the same best case as TarjanSafe and remains polynomial in the worst case. PI is well-known as an MDP algorithm (Howard 1960), and arguably, successful goal-reaching FOND algorithms such as PR2 (Muise, McIlraith, and Beck 2024) fit within its framework. In our setting, the idea of PI is that it starts with some policy π, determines where π is unsafe, fixes π, and repeats until π is safe or s_I is proven unsafe. We define nPI using a Value Function V with

Algorithm 1: Naïve Policy Iteration

```
1  Function nPI(Θ, s_I, π_init)
2      V(s) ← [[s ∈ S_f]] ∀s ∈ S
3      π ← π_init
4      repeat
5          V, saw-unsafe ← Mark-Unsafe(Θ, s_I, V, π)
6          if ¬saw-unsafe or V(s_I) = 1 then
7              return V(s_I)
8          π ← Greedy(Θ, s_I, V)
9  Function Mark-Unsafe(Θ, s_I, V, π)
10     E ← DFS post order of policy graph Θ^π from s_I
11     for s ∈ E do
12         V(s) ← min_{a∈A(s)} Q(s, a)
13     saw-unsafe ← [[∃s ∈ E s.t. V(s) = 1]]
14     return V, saw-unsafe
15 Function Greedy(Θ, s_I, V)
16     new π undefined everywhere
17     while π has undefined state reachable from s_I do
18         s ← undefined state reachable from s_I
19         π(s) ← argmin_{a∈A(s)} Q(s, a)
20     return π
```

the semantic that $V(s) = 1$ if we know that s is unsafe, and $V(s) = 0$ if we are unsure. Initially, we only know that the fail states are unsafe so $V(s) \leftarrow [\![s \in \mathsf{S}_f]\!]$ (line 2). We use the shorthand $Q(s, a) = \max_{s' \in \mathcal{T}(s,a)} V(s')$, which is 1 if we know it is unsafe to apply a in s, and 0 if we are unsure. Then, we propagate the states that we know are unsafe by updating $V(s)$ with its minimal Q-value (line 12).

Theorem 4. *nPI has a best-case runtime of $\Theta(\pi_{\text{min-safe}})$ (if a safe policy exists).*

Proof sketch. Similar argument as for thm. 1.

Theorem 5. *nPI has a worst-case runtime of $O(bmn)$.*

Proof sketch. In each call to *Mark-Unsafe*, if π remains unsafe, then at least one state with $V(s) = 0$ gets updated with $V(s) \leftarrow 1$. Thus, in the worst case, nPI updates a single $V(s)$ per iteration and then terminates. This gives $O(n)$ iterations with an $O(bm)$ DFS in each pass.

This nPI has various inefficiencies which we address in the improved iPI algorithm. First, the construction of a greedy policy and its evaluation are combined. This means the candidate policy's envelope only needs to be traversed once per iteration, and it enables us to stop the construction of the greedy policy as soon as an unsafe state has been detected. Second, instead of π_{init}, iPI uses an action order $\mathbb{A}$ where $\mathbb{A}(s)$ returns the same actions as $\mathsf{A}(s)$ but ordered. Consequently, iPI no longer constructs policies that are greedy w.r.t. V, but rather constructs policies according to the preferences of $\mathbb{A}$, which is faster in our setting. The choice of $\mathbb{A}$ can be important, since a good ordering may quickly lead to a safe policy or to a way to prove s_I as unsafe, and a bad one causes iPI to waste time. The best-case and worst-case runtimes of iPI are the same as nPI, by similar arguments. Our technical report describes iPI in detail.

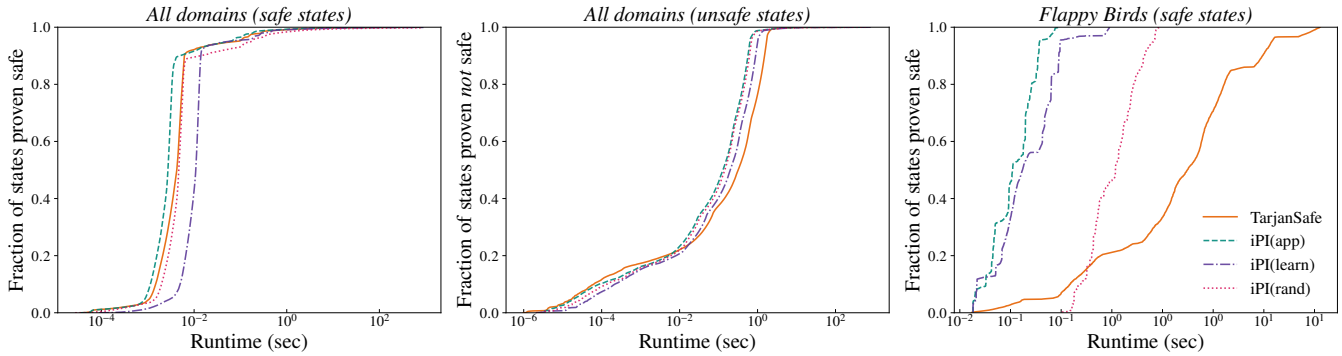

Figure 2: Cumulative plots of the proportion of states decided w.r.t. time. The plots separate safe and unsafe states: (left) is an aggregation over all domains' safe states, (middle) is over all unsafe states, and (right) is over safe states in Flappy Bird.

4 Experiments

We empirically compare iPI with JEA's TarjanSafe for deciding state safety: given a learned policy π that we wish to check for safety, we find a state s in π's graph, and interrogate iPI and TarjanSafe whether s is safe. Recall that finding such states is at the core of JEA's pipeline for testing the safety of the learned policy. We consider iPI with three different action orderings (1) *app*: the order of the model (same as TarjanSafe); (2) *learn*: using the learned action policy that we are evaluating; (3) *rand*: randomised. We do not consider Prop-$\mathcal{U}$ because it runs out of memory. Our implementation extends JEA's C++ code and is available at (Schmalz and Jain 2026). All experiments were run on Intel Xeon E5-2660 CPUs with 12GB memory and a timeout of 15 minutes.

Benchmarks. We use JEA's benchmarks (scaled up), consisting of non-deterministic variants of *blocksworld* and *transport*; deterministic control benchmarks (*bouncing ball*, *follow car* and *inverted pendulum*); as well as different variants of a transport-like domain called *one/two-way line*, where a truck moves uni-(respectively bi-)directionally on a line, carrying packages to the right. We also introduce a new benchmark, *Flappy Bird*. A safe policy directs the bird around a repeating track indefinitely while avoiding stationary obstacles. When safe policies exist, this showcases TarjanSafe's bad performance, taking inspiration from the example in fig. 1 in a more practical setting. Following JEA, all problems are encoded in JANI (Budde et al. 2017). For learned policies to test, we trained neural networks via Q-learning with two hidden layers of size 64.

Results. Our results are summarised in fig. 2. We note the benchmarks are solved in small timescales: the iPI variants solve almost all problems within 1 sec. This is not because the problems are small: we handle instances with up to 80 integer variables bounded by 70. To motivate this further, Prop-$\mathcal{U}$ runs out of memory as it enumerates all exponentially-many fail states (w.r.t. JANI description); similarly, Prop-$\mathcal{U}$ with regression over Binary Decision Diagram (BDDs), which can represent a set of states compactly, also runs out of memory as the BDDs become too large. Rather, the benchmarks are solved quickly by exploiting the features of state-avoiding problems. For deciding safe states, there are often cycles that TarjanSafe and iPI use to quickly construct safe policies that avoid fail states. For deciding unsafe states, TarjanSafe and iPI do not need to explore deeply before finding a way to propagate unsafety to s_I.

Comparing the variants of iPI: iPI(app) dominates the others. The orderings themselves do not appear to have a significant impact: all orderings tend to guess safe policies quickly (for safe states) and propagate the unsafe region to s_I efficiently (for unsafe states). Rather, the performance difference comes from implementation, namely the overhead of sorting with *learn* and randomising with *rand*.

Comparing iPI to TarjanSafe: For deciding states over all domains (left and middle), there is no significant difference between the approaches. These benchmarks tend to be amenable to TarjanSafe, so iPI indeed has similar best-case behaviour to TarjanSafe. For deciding safe states in Flappy Birds, iPI scales exponentially better than TarjanSafe (note the runtime scale is logarithmic). This confirms our theory that iPI is exponentially faster than TarjanSafe in its worst case. So, iPI never performs significantly worse than TarjanSafe, and in some cases performs exponentially better.

Summary. These results confirm our theory that iPI(app) dominates TarjanSafe in the sense that it is similar for problems amenable to TarjanSafe, and exponentially better in problems that are problematic for TarjanSafe such as Flappy Bird. Furthermore, the results show that within iPI, the default action ordering performs best in our setting. We give a breakdown over domains and more discussion in the technical report, these are consistent with our findings here.

5 Conclusion

For the state-safety decision problem (a necessary component of JEA's safety assurance pipeline) we considered JEA's TarjanSafe algorithm, and introduced Prop-$\mathcal{U}$ and iPI. We showed TarjanSafe has an exponential worst-case runtime but a good best case; in contrast, Prop-$\mathcal{U}$ has a linear worst case but impractical best case; iPI fills the gap, combining a polynomial worst case while maintaining the same best case as TarjanSafe. This places iPI as the current best algorithm for deciding state safety in our framework.

Acknowledgements

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – GRK 2853/1 "Neuroexplicit Models of Language, Vision, and Action" - project number 471607914.

References

Budde, C. E.; Dehnert, C.; Hahn, E. M.; Hartmanns, A.; Junges, S.; and Turrini, A. 2017. JANI: Quantitative Model and Tool Interaction. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 151–168.

Cimatti, A.; Pistore, M.; Roveri, M.; and Traverso, P. 2003. Weak, Strong, and Strong Cyclic Planning via Symbolic Model Checking. *Artificial Intelligence*, 147(1–2): 35–84.

Daniele, M.; Traverso, P.; and Vardi, M. Y. 2000. Strong Cyclic Planning Revisited. In *Recent Advances in AI Planning*, 35–48.

Diekert, V.; and Kufleitner, M. 2022. Reachability Games and Parity Games. In *Int. Colloquium on Theoretical Aspects of Computing (ICTAC)*.

Howard, R. A. 1960. *Dynamic Programming and Markov Processes*. Technology Press of Massachusetts Institute of Technology.

Jain, C.; Sherbakov, D.; Vinzent, M.; Steinmetz, M.; Davis, J.; and Hoffmann, J. 2025. Policy Safety Testing in Non-Deterministic Planning: Fuzzing, Test Oracles, Fault Analysis. In *Proc. of European Conf. on Artificial Intelligence (ECAI)*.

Klauck, M.; Steinmetz, M.; Hoffmann, J.; and Hermanns, H. 2018. Compiling Probabilistic Model Checking into Probabilistic Planning. In *Proc. of the Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 150–154.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M. A.; Fidjeland, A.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-Level Control Through Deep Reinforcement Learning. *Nature*, 518(7540): 529–533.

Muise, C.; McIlraith, S. A.; and Beck, J. C. 2024. PRP Rebooted: Advancing the State of the Art in FOND Planning. In *Proc. of AAAI Conf. on Artificial Intelligence*, 20212–20221.

Schmalz, J.; and Jain, C. 2026. Code and Data for Algorithms for Deciding the Safety of States in Fully Observable Non-deterministic Problems. URL: https://doi.org/10.5281/zenodo.18926835.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature*, 529: 484–503.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T.; Simonyan, K.; and Hassabis, D. 2018. A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go Through Self-Play. *Science*, 362(6419): 1140–1144.

Tarjan, R. 1972. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2): 146–160.

Toyer, S.; Thiébaux, S.; Trevizan, F.; and Xie, L. 2020. ASNets: Deep Learning for Generalised Planning. *Journal of Artificial Intelligence Research*, 68: 1–68.