ASP-Based Probabilistic Policy Fixing for Norm Compliant RL

Sebastian Adam , Thomas Eiter
Institute of Logic and Computation, Vienna University of Technology
{sebastian.adam, thomas.eiter}@tuwien.ac.at

Abstract

Reinforcement learning (RL) is commonly used to learn reward-optimizing policies. However, RL policies are not always trained with ethical behavior in mind, which can lead an agent to violate social or legal norms in pursuit of its goal. Retraining agents with additional norms is not always feasible, especially in complex stochastic environments. To mitigate this issue, we present a probabilistic policy fixing framework that adapts norm-agnostic policies online. Using Answer Set Programming (ASP), we generate policy fixes that minimize deviations from the RL policy while optimizing for norm adherence against a set of sampled worlds. Based on the Rule of Three and Hoeffding's inequality, we provide guarantees that fixed policies are near optimal, given a specified level of confidence.

1 Introduction

Reinforcement Learning (RL) techniques have proven to be successful in training autonomous agents for tasks like mastering complex games [Silver *et al.*, 2016; Berner *et al.*, 2019] or manipulating robotic controls [Gu *et al.*, 2017]. However, a high-reward policy is not necessarily compliant with social, legal or ethical norms. In fact, the most efficient path to a goal often involves risky or prohibited behavior, creating a conflict between reward maximization and norm compliance. For instance, a delivery drone minimizing flight time may choose to traverse a restricted no-fly zone rather than taking a safer, although longer, detour [Dai *et al.*, 2025].

Ensuring norm compliance in safety-critical domains is an important and non-trivial problem. However, traditional remediation strategies, such as retraining the agent upon discovering a norm violation can be computationally expensive and too slow for online adaptation. While methods like shielding [Könighofer *et al.*, 2025] exist to block unsafe actions, they typically enforce hard constraints (safe vs. unsafe). This binary approach is not suited for deontic norms [Meyer *et al.*, 1994], where total compliance may be impossible and the agent must remain operational and instead minimize norm violations triggered by his actions.

In stochastic domains, the challenge is even bigger. Strict adherence to norms in every possible future scenario (i.e., optimizing for the worst case as in [Adam and Eiter, 2025]) can lead to overly conservative behavior, while exact probabilistic planning often requires complex logic programming with high computational complexity [Littman *et al.*, 1998].

To address this issue, we propose a framework[1] for ad-hoc, probabilistic policy fixing that combines the strengths of RL and logic programming. Our approach treats norm-agnostic RL policies as a baseline that can be dynamically overridden by an emergency plan (i.e., a *policy fix*) when a violation risk is detected. Rather than computing these fixes by model counting for exact probabilities, we employ the following pipeline: a monitoring program continuously evaluates the agent's current trajectory against a set of sampled world evolutions, based on an exact model of the environment, triggering an emergency planning phase only when predefined safety guarantees cannot be met. By utilizing Answer Set Programming (ASP) [Brewka *et al.*, 2011] to check norm violations and search for alternative policies over this finite set of sampled worlds, we enable the agent to find a path that minimizes norm violations while remaining as faithful as possible to the original reward-maximizing objective.

This approach allows the agent to handle deontic norms in stochastic domains, enabling emergency planning and minimizing norm violations even when total compliance is impossible. By leveraging RL preferences and ASP optimization within this framework, we are able to compute high quality norm compliant alternatives to a RL policy for a finite horizon. Specifically, we make the following contributions:

- **Statistical Model Checking Intervention.** We use sampling to predict violation probabilities over a finite horizon. By leveraging the Rule of Three and Hoeffding's inequality, we provide formal guarantees on the optimality of our computed policies.
- **ASP-Based Emergency Planning.** When a safety threshold is breached, the framework invokes an ASP planner to generate an emergency policy fix. This plan is optimized against sampled trajectories to minimize expected norm violations respecting the agent's original reward-seeking behavior.
- **Computational Efficiency.** By planning against sets of sampled world evolutions rather than embedding probabilistic semantics directly into the logic, we maintain the efficiency

[1] Implementation, benchmarks, and further supplementary material are available at https://gitlab.tuwien.ac.at/sebastian.adam/asprob

of standard deterministic ASP solvers. This avoids the complexity of exact probabilistic model counting, while producing plans that are near optimal with a specified confidence.

- **Norm Complexity.** By leveraging disjunctive ASP without expensive encodings like the saturation technique, we are able to encode and check for a variety of different norms. Notably these include timed and contrary-to-duty norms as well as exceptions in addition to maintenance norms.

Our experiments with an implementation using the sota ASP solver *clingo* show that probabilistic policy fixing can be a successful interim solution, tweaking a RL policy slightly to significantly improve norm compliance. Finally, checking norm compliance with our framework using ASP improves the transparency and explainability compared to solutions relying exclusively on RL.

2 Related Work

In shielding, the learning objective of an RL agent is decoupled from its safety constraints by employing a logical monitor (i.e., a "shield") that blocks unsafe actions at runtime. Shielding architectures are generally categorized into (1) pre-shielding, where the shield provides the agent with a list of valid actions to choose from, and (2) post-shielding, where the shield monitors the agent's selected action and overwrites it if a violation is expected [Könighofer *et al.*, 2025].

Our framework is therefore more closely related to post-shielding, as we check proposed actions of a RL policy and intervene only when necessary. However, our approach differs in three critical aspects. First, shields are typically already active during the training phase, whereas we assume a norm-agnostic policy. Second, while standard shielding enforces binary hard constraints (safe vs. unsafe), we operate on deontic norms. Unlike hard constraints, norms allow for prioritized violations when total compliance is impossible (e.g., minimizing the severity of a violation in an emergency), rather than simply blocking an action. Third, while probabilistic shields exist, they typically rely on expensive formal model checking or state augmentation [Könighofer *et al.*, 2025; le Court *et al.*, 2025]. In contrast, we use sampling techniques to effectively estimate norm violation probabilities without the scalability bottlenecks of formal logic.

Extensions of ASP handling uncertainty typically embed probabilistic semantics directly into the logic. Frameworks such as P-log [Baral *et al.*, 2009] and LP$^{\text{MLN}}$ [Lee and Wang, 2016] assign probabilities to rules or atoms, requiring specialized inference mechanisms like *plingo* [Hahn *et al.*, 2025] to compute expected values or most probable worlds. Credal semantics [Cozman and Mauá, 2020] offer a more general approach, representing uncertainty through probability bounds, but yield lower/upper probabilities rather than a single expected value. In the context of safe RL, DARLING [Leonetti *et al.*, 2016] utilizes standard ASP to constrain RL exploration but assumes a deterministic domain model for planning. Our approach occupies a middle ground: we retain the efficiency of standard, deterministic ASP solvers, but we take stochastic uncertainties into account by planning against a set of sampled world evolutions. This avoids the complexity of probabilistic logic programming, where model counting is $\# \cdot$ co-NP-complete for disjunctive programs [Fichte *et al.*, 2017] and typically requires approximation [Kabir *et al.*, 2022] anyway.

Differentiable SAT/ASP [Nickles, 2018] introduces a multi-model optimization approach, where the solver iteratively samples a multi-set of models to minimize a cost function defined over statistical properties (e.g., atom frequencies), utilizing gradient descent to guide branching decisions. While sampling is reminiscent of our approach, a fundamental objective gap distinguishes it from our work: Diff-ASP aims to approximate a target distribution across many models (e.g, generating a set of possible worlds where a specific event occurs with 30% probability). In contrast, policy fixing searches for a single emergency plan for the agent's current state.

Our probabilistic policy fixing framework is closely related to the normative supervisor by Neufeld *et al.* [2021] and the emergency planning framework by Adam and Eiter [2025]. While the normative supervisor checks policies against a set of norms and hinders the agent from executing actions leading to an immediate norm violation, Adam and Eiter [2025] expand on this idea and check whether a policy will inevitably lead to a norm violation within a fixed horizon. Both of these methods, however, only consider simple maintenance norms.

For dealing with temporal and generally more complex norms, [Neufeld *et al.*, 2024] introduced a technique to shape the reward of a RL agent using so-called *restraining bolts* [De Giacomo *et al.*, 2019]. A normative variant of the restraining bolts, expressed in LTLf [De Giacomo and Vardi, 2013] is used to minimize penalties for norm violations. As our proposed framework works as an ad-hoc "emergency" solution for an existing RL policy, a combined application is conceivable: While we temporarily fix norm violating behavior with our framework, a norm-abiding RL policy is trained using the restraining bolt technique.

3 Preliminaries

Reinforcement Learning. RL is a machine learning approach in which an agent learns to make decisions by interacting with an environment, receiving rewards for desired behavior [Sutton and Barto, 2018]. We formally model this interaction as a Markov Decision Process (MDP), defined by the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, s_0, \mathcal{P}, \mathcal{R} \rangle$. Here, $\mathcal{S}$ is the state space, $\mathcal{A}$ the set of possible actions, $s_0 \in \mathcal{S}$ is an initial state, $\mathcal{P}(s'|s,a)$ represents the transition probability to state s' given action a in state s and $\mathcal{R}(s,a)$ is a reward function providing immediate feedback. The agent's goal during training is to find an optimal policy π, maximizing the expected cumulative reward (discounted by $\gamma \in [0,1)$).

In this work, we focus on value-based RL algorithms and specifically Q-learning [Watkins and Dayan, 1992], where the agent learns an action-value function $Q(s,a)$. This function represents the expected utility of taking action a in state s and following the learned policy π thereafter. To handle the high-dimensional state spaces typical of complex environments, we utilize linear function approximation. Instead of a simple tabular representation (viz. Q-table), the Q-values are approximated as $Q(s,a) \approx \phi(s,a)^\top \mathbf{w}$ where $\phi(s,a)$ is a feature vector extracted from the state-action pair, and $\mathbf{w}$ is a vector of learnable weights. This allows the agent to generalize learned

behaviors across similar states by updating $\mathbf{w}$.

While we focus on standard Q-learning, our approach is not restricted to this specific algorithm. It can be applied to deep variants of Q-learning like DQN [Mnih *et al.*, 2015] and in general every value-based RL method (and even non-RL policies), provided the availability of an action-value function.

Norms. Norms, as described in many deontic logics such as Standard Deontic Logic (SDL) [Gabbay *et al.*, 2013], characterize a behavior p as obligatory $\mathbf{O}(p)$, permitted $\mathbf{P}(p)$, or prohibited $\mathbf{F}(p)$. Commonly, $\mathbf{O}$ is taken as the primitive operator and the others are defined using negation $\neg$ as follows:

$$\mathbf{P}(p) := \neg\mathbf{O}(\neg p), \qquad \mathbf{F}(p) := \mathbf{O}(\neg p).$$

While standard approaches to safe Reinforcement Learning often rely on hard constraints to prune the action space, such methods can be overly restrictive when dealing with deontic norms. In contrast, deontic norms usually allow some kind of violability without rendering the resulting state impossible.

A *normative system* is a set of norms, with varying levels of importance, priority and complexity. In addition to *maintenance obligations* [Governatori *et al.*, 2007], which require the agent's continuous adherence throughout execution, we consider more complex structures like *Contrary-to-Duty* (CTD) obligations [Prakken and Sergot, 1996]. The latter are particularly relevant for autonomous agents as they specify secondary duties that arise only after a primary norm has been violated, providing a recovery logic for sub-ideal states. ASP has proven effective to handle such complex normative systems across various scenarios [Hatschka *et al.*, 2023; Governatori, 2024]. Furthermore, normative extensions to ASP, such as the recently introduced *deolingo* [Cabalar and Manteiga, 2025] extension for the *clingo* solver provide a convenient method to represent deontic concepts alongside standard ASP, while retaining reducibility to ordinary ASP.

Answer Set Programming. ASP is a declarative programming paradigm rooted in the stable model semantics of logic programs [Brewka *et al.*, 2011]. A logic program consists of rules of the form:

$$a_1 \mid \ldots \mid a_l \leftarrow b_1, \ldots, b_m, not\, c_1, \ldots, not\, c_n \qquad l, m, n \geq 0$$

where atoms $a_{1..l}$ make up the *head* and atoms $b_{1..m}$ combined with negated atoms $c_{1..n}$ form the *body*. A rule with an empty head ($l = 0$) is a hard constraint, while rules with multiple head atoms ($l > 1$) are disjunctive rules.

The default negation "*not*" (negation as failure) introduces nonmonotonicity, distinguishing ASP from monotonic logics such as classical propositional satisfiability (SAT). This allows ASP for modeling defaults and exceptions, where conclusions can be withdrawn in the light of new information. In addition, advanced ASP solvers such as *clingo* [Gebser *et al.*, 2014] and *DLV* [Alviano *et al.*, 2017] allow for using weak constraints. While standard constraints prune the search space, weak constraints penalize answer sets if their body is satisfied, making them the ideal choice for encoding deontic norms.

4 Probabilistic Policy Fixing

Adam and Eiter [2025] defined the notion of a *(k-) policy fix* for a norm violating policy π. Intuitively, a k-policy fix minimizes the worst-case norm violations over a horizon k, while ensuring minimal deviation from π. While geared to the worst-case scenario, it is potentially overly conservative in stochastic environments and relies on solving conformant planning, which is EXPSPACE-complete [Rintanen, 2004] in general and already Σ_2^P-complete (cf. [Baral *et al.*, 2000]) for plans over a polynomial horizon k, assuming transitions are guaranteed or checking whether a follow-up state exists can be done in polynomial time.

To address this, we introduce *probabilistic policy fixing*. We relax the requirement of universal compliance to ensuring that the probability of a norm violation remains below a given threshold with high confidence.

Let $\mathcal{T}$ be the set of all trajectories starting at time t given a policy π. We define a binary violation indicator function $V : \mathcal{T} \to \{0, 1\}$ such that:

$$V(\tau) = \begin{cases} 1 & \text{if trajectory } \tau \text{ violates the norm } \Phi, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Instead of ensuring $\sum_{\tau \in \mathcal{T}} V(\tau) = 0$ as strict policy fixes in [Adam and Eiter, 2025] do, we aim for a policy π' maximizing the expected reward R of a policy from the set Ψ of all policies that minimize the expected value of a norm violation V.

Definition 1. *Given a state s, a policy π, its reward function R and the indicator function V, a* probabilistic policy fix *for π is a policy π' such that*

$$\begin{aligned} \pi' &= \operatorname{argmax}_{\pi'' \in \Psi} \mathbb{E}_{\tau \sim \pi''}[R(\tau)] \quad \textit{where} \\ \Psi &= \operatorname{argmin}_{\pi'' \in \Pi} \mathbb{E}_{\tau \sim P(\cdot|\pi'', s_t)}[V(\tau)]. \end{aligned} \tag{2}$$

As computing exact probabilities is often intractable, we approximate them by sampling. We generate N independent trajectories $\tau_1, \ldots, \tau_N$ with finite horizon h by simulating the environment under π''. The empirical violation probability is

$$\hat{p} = \textstyle\sum_{i=1}^{N} V(\tau_i)/N. \tag{3}$$

As $V(\tau)$ is a bounded variable in $[0, 1]$, we use the Rule of Three (RoT) [Jovanovic and Levy, 1997] and Hoeffding's inequality [Hoeffding, 1963] to bound the difference between the estimated probability $\hat{p}$ and the true probability p.

The RoT tells us that $3/n$ is an upper 95% confidence bound for our binomial distributed probability p, if in n independent samples no norm violations occur. This rule is derived from the fact that the true probability p of observing zero norm violations is $(1 - p)^n$. Assuming p should be no greater than an error tolerance ϵ with confidence $1 - \delta$, we thus can set:

$$(1 - \epsilon)^n = \delta. \tag{4}$$

This leaves us with a general formula that we can solve for n:

Lemma 1. *Given a policy π', from*

$$n = \ln(\delta)/\ln(1 - \epsilon) \tag{5}$$

randomly sampled violation-free worlds (i.e., trajectories) we can conclude with confidence $1 - \delta$ that π' will not violate norms, up to ϵ-error rate.

Example 1. *To assure with 95% confidence that a policy fix will not violate the norm with probability of more than 5% the RoT tells us that we need $n = 60$ independent sample worlds without a norm violation.*

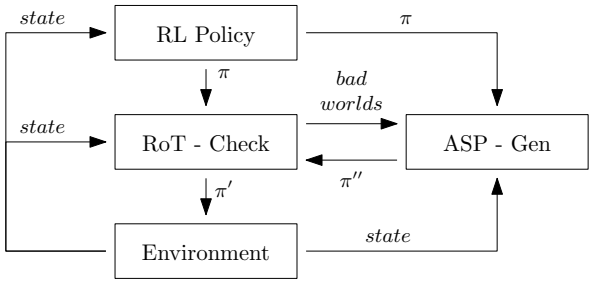

Figure 1: ASP Probabilistic Policy Fixing Framework

In our framework, the RoT is used as a practical tool to calculate the number of samples needed, to evaluate a candidate policy on its norm compliance.

However, in some scenarios a norm compliant policy is not possible. Then RoT is not helpful, as it not readily extends to cases where the set of sampled worlds contain norm violations. For such cases, we use another statistical bound, viz. the *Hoeffding's inequality* [1963], which gives a more general bound on the probability that $\hat{p}$ differs from p by more than ϵ:

$$P(|\hat{p} - p| \geq \epsilon) \leq 2e^{-2N\epsilon^2}. \quad (6)$$

However, as we will use the calculated number of policies using the inequality to evaluate multiple policies, we must ensure that the estimates are accurate for all policies in Π simultaneously. By applying the union bound over the set of candidates [Mohri *et al.*, 2012], the probability that the estimation error exceeds ϵ for *any* policy is bounded as follows:

$$P(\exists \pi'' \in \Pi : |\hat{p}_{\pi''} - p_{\pi''}| \geq \epsilon) \leq |\Pi| \cdot 2e^{-2N\epsilon^2}. \quad (7)$$

The required number N of samples is thus as follows.

Lemma 2. *To guarantee with confidence $1 - \delta$ that a policy π' is within error 2ϵ optimal w.r.t. norm violations over all candidate policies,*

$$N \geq (\ln 2 + \ln |\Pi| - \ln \delta)/2\epsilon^2 \quad (8)$$

random sample worlds from π' are sufficient.

Example 2. *If the agent can choose any of five actions over a horizon of 3, the number $|\Pi|$ of potential policies is 5^3. To assure with 95% confidence that the probability of norm violation in a policy optimized against sampled worlds is within 5% of the theoretical optimal policy, Hoeffding's inequality tells us that we need $N = 1704$ sample worlds.*

5 ASP Framework

We present a framework that computes a probabilistic policy fix for a RL policy using two ASP programs and a set of randomly sampled worlds. The numbers of sampled worlds required n and N are calculated before invoking the framework according to the RoT and Hoeffding's inequality respectively, given a confidence level $(1 - \delta)$ and an error tolerance ϵ.

The proposed framework, pictured in Figure 1 and outlined in Algorithm 1, augments a traditional RL execution loop. Instead of naively executing the policy π, it serves as input to an ASP program that checks whether π survives a norm-compliance check over a finite horizon h in n world evolutions,

Algorithm 1 Probabilistic Policy Fixing

Input: state s, policy π_q, conf. $1 - \delta$, error tol. ϵ, horizon h
Output: Probabilistic policy fix π'

```
Compute n, N from RoT (5) & Hoeffding (8) given δ, ε.
Let π'' = π_Q, c = 0 and Π = ∅.
while c · n < N and π'' ∉ Π do
    Set Π = Π ∪ π'' and c = c + 1
    Let v be the set of violating worlds received from RoT-Check on π'' using n fresh sample worlds.
    if v is empty then
        return π''
    else
        Set π'' according to ASP-Gen using π_Q and all bad world samples extended by v as input
    end if
end while
Set π'' according to ASP-Gen using π_Q and the full N world samples as input
return π''
```

sampled under π. If the policy survives the check, its next proposed action is executed unaltered in the environment.

In case π does not survive the initial check, we enter the loop depicted in Figure 1 between the two ASP programs labeled "RoT-Check" and "ASP-Gen". RoT-Check's answer sets include information about all "bad worlds" in which a candidate policy violates a norm and ASP-Gen uses this information to plan a new policy for h steps which minimizes norm violations in all bad worlds with high priority and maximizes the expected reward according to π with low priority. This new policy π'' is again tested using the RoT, but with a fresh batch of n sample world evolutions under π''. We repeat this procedure until one of the following stopping criteria is met:

(1) We identify a policy that survives the RoT check
(2) ASP-Gen repeats a previously tested policy
(3) The number of all worlds used in RoT-Check exceeds N

If we are able to identify a policy π' that survives the RoT check (1), we can say with confidence $1-\delta$ that the probability of a norm violation following that policy does not exceed ϵ and hence only differs at most ϵ from a theoretical optimal policy with 0% norm violation probability.

However, a policy with a probability $\leq \epsilon$ for norm violations may simply not be possible. In such cases, ASP-Gen often repeats a previously computed policy (2), as it converges to a policy minimizing norm violations in bad worlds. In this case, chances of identifying a policy that survives by continuing the loop are negligible. Hence, we stop the loop and provide the full N samples to the generator program, which will return the policy π' that at least minimizes the probability of a norm violation $\hat{p}_{\pi'}$ in the sampled worlds. As the number N was calculated using Hoeffding's inequality, we can say with confidence $1 - \delta$ that $|p_{\pi'} - \hat{p}_{\pi'}| \leq \epsilon$.

Furthermore, as we are using the inequality in combination with the union bound and ASP-Gen now considers every policy $\pi'' \in \Pi$, we can say with confidence $1 - \delta$ that the difference between $p_{\pi'}$ and the norm violation probability of a theoretical optimal policy $p_{\pi_{opt}}$ is bounded by 2ϵ.

In (3) the number of sampled worlds already exceeds N. Hence, we stop the loop and use the final policy π' computed by ASP-Gen using the N world samples, similar to case (2).

Summarizing, based on Lemmas 1 and 2, we can derive

Proposition 1. *Alg. 1 outputs, given a state s, a policy π_Q, and parameters $1-\delta$, ϵ, h with confidence $1-\delta$ a policy fix π' that is up to h steps optimal w.r.t. norm violations under error bound 2ϵ.*

ASP programs. The majority of rules encoding the model of a domain is shared between ASP-Gen and RoT-Check. However, ASP-Gen contains a disjunctive (guessing) rule for the agent's actions and optimization statements, enabling the solver to search for an optimal policy. In contrast, RoT-Check takes a fixed policy of the agent as input, merely searching for worlds in which the policy leads to a norm violation.

In the following, we highlight key parts of the ASP implementation for the domain detailed in Section 6 using simplified rules, while the full programs can be found in our repository.[1]

Sample World Evolutions. Rules that rely on probabilities contain an atom carrying a random number $\texttt{R} \in [0, 100)$ in their body. This number, representing the random luck of a stochastic event, is sampled externally and given to both ASP programs. This avoids dependence on randomness of an ASP solver[2] and enables that sample values of a single world manifest as different concrete evolutions, depending on the environment, considered by the ASP implementation.

One of the frog movement rules of the domain detailed in Section 6 serves as an example:

```
action(AP,F,T,W) :- frog(X,Y,F,T,W),
                    pref(X,Y,T,AP),
                    rnd(F,T,W,R), R>=0, R<70.
```

Here, `rnd` provides the random variable `R` and the frog `F` at time `T` takes the preferred action `AP` for position `X,Y` iff $\texttt{R} \in [0, 70)$. The agent's actions directly influence the preferred actions of the frogs at certain positions, hence depending on the policy, the concrete derived action `AP` may change.

RL preference. A key aspect of our framework is the capability to retain information of the RL policy, even when we deviate from the optimal path. To provide ASP-Gen with information about the expected reward of the RL policy, given the current state s and an action a, we use clingo's capabilities to interact with a Python program [Gebser *et al.*, 2014].

The atom `reward(@qvalue(G,P,D_G,D_P),T)` is a concrete example of such an implementation, containing all relevant information also found in the feature vector described in Section 6. The features `G,P,D_G,D_P`, describing immediate consequences and distances, are computed within the ASP program based on the model of the environment for each time point `T`, given the agent's policy. Then `@qvalue` calls the equivalently named Python function, which in turn queries the Q-function and returns the calculated value accordingly.

Norms. In our framework, each sampled world evolution is either norm compliant or norm violating. We do not grade norm violations given some kind of prioritization or the time step at which they occur. Hence, any norm violation in a sampled world `W` leads the solver to derive the atom `viol(W)`. An example encoding for the simple maintenance norm (MTN1), described in Section 6, is as follows:

```
viol(W) :- capture(F,T,W).
```

`capture` is an atom derived if the agent and the frog `F` are on the same cell at the same time. As both ASP programs check/optimize against concrete world evolutions, we are, however, not limited to such simple maintenance norms. By adding atom `ctd(F,T,W)`, which is active for four time steps after the agent violates the second maintenance norm (MTN2) we can track contrary-to-duty norms like this:

```
viol(W) :- ctd(F,T,W), not ctd_c(F,W).
```

Here we use default negation to trigger the norm violation, if (MTN2) was triggered and our agent is not able to recover by capturing the frog, causing the solver to derive `ctd_c`.

In addition, we can use default negation to allow exceptions to a norm; e.g., the exception for (MTN1) can be encoded as

```
viol(W) :- capture(F,T,W), not ctd(F,T,W).
```

In RoT-Check `viol(W)` merely marks violating worlds `W`, whereas ASP-Gen uses the weak constraint `:~ viol(W). [1@1, W]` to minimize its occurrences in the answer set.

ASP Complexity. RoT-Check is a plain ASP program using default negation but no disjunction, as the agent's actions and the randomness of the environment are given as input. For ASP-Gen, we rely on choice rules and optimization by the ASP solver. Hence, checking for norm violations is in NP, while generating a new policy is in FP^{NP}, i.e., feasible in polynomial time with an NP oracle, cf. [Leone *et al.*, 2006]. This offers an advantage over policy construction with saturation techniques involving exact model counting with complexity $\#\cdot$co-NP, which is not known to be feasible in FP^{NP}.

6 Experiments

In this section, we show the applicability of our framework on an exemplary stochastic domain in a 2-dimensional grid game setting. This domain is carefully crafted to be representative as a benchmark for normative compliance in stochastic grid games. For increased complexity, the environment contains random elements that are (in)directly reactive to the agent's actions. Hence, naive adaption based on observed norm violating behavior will not yield a viable policy.

Our focus in testing lies on the performance and norm compliance of our agent in this environment, where he is subject to increasingly more complex norms. For evaluation, we compare the norm compliance of four different policies:

- **RL** acts according to the (norm agnostic) RL policy π_Q, trained using Q-learning
- **PPF** employs our probabilistic policy fixing framework starting every step with π_Q in the first RoT check
- **PPF*** extends PPF by caching previously computed policy fixes and uses them, extended with fresh actions of π_Q up to horizon h, in the first RoT check
- **UPF** employs the utility-based policy fix, presented in [Adam and Eiter, 2025] with a window radius of 10

[2]Answer sets enumerated by ASP solvers are in reality not independent of each other, as they rely on methods like backtracking.

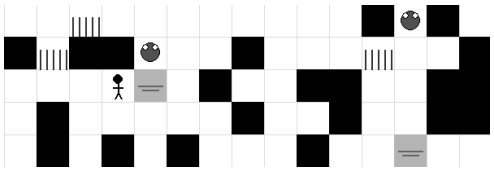

Figure 2: Example instance of *Gardener* game

Policy fixing frameworks are tested with horizons $h = 3$ and $h = 5$. Both probabilistic versions are tested using multiple combinations of confidence and error tolerance, ranging from $60-99\%$ and $25-3\%$, respectively. Each test is run 100 times on randomly generated instances of the same size, where 1%, 2%, 4% and 30% of the cells are covered with frogs, puddles, grass patches, and walls respectively (an excerpt of such an instance is shown in Fig. 2). We used the same random seeds to generate the instances for the different algorithms. We measure the number of steps needed, how often a framework deviates from the RL policy, compute time for policy fixes and how often each norm is violated. We expect to confirm that:

- **(H1)** While PPF/PPF* deviates from π_Q to comply with norms, it needs less further steps to finish an instance than UPF, also resulting in less deviations from π_Q.
- **(H2)** PPF/PPF* reduces the average compute time compared to UPF, as it does not rely on a co-NP check.
- **(H3)** PPF* reduces the average compute time compared to PPF, as norm-compliant trajectories are cached.
- **(H4)** PPF/PPF* enables enforcement of multiple and more complex norms compared to UPF.

In the following, we highlight exemplary results, while the full testing results can be found in our repository.[1]

Domain. As a benchmark domain, we consider an extension of the *Gardener* environment in [Adam and Eiter, 2025]. In our variant of the game, there is no target cell, as seen in the instance shown in Fig. 2. Instead, the agent tries to reach a score of 300. The agent is awarded 10 points when they mow a grass patch (i.e. steps onto a grass patch) and 5 points when they drain a puddle (i.e. steps onto one of the four orthogonal neighbors of a puddle). Mowing a grass patch (vertical lines in Fig. 2) or draining a puddle (horizontal lines in Fig. 2) leaves the cell inactive for the next 50 respectively 20 steps, after which it resets. In addition to the actions $up, right, down, left$, moving the agent one cell into the according direction, our agent may $stay$ in the current cell.

Each instance contains frogs (circles in Fig. 2) inhabiting the garden. Frogs move randomly each step into one orthogonally neighboring cell and the agent captures a frog if they both inhabit the same cell simultaneously. In contrast to the original *Gardener* game, frogs move not uniformly at random to a neighboring cell but more likely (70%) favor jumping onto one that brings them closer to the nearest full puddle. Finally, if a frog is on any of the eight cells surrounding a puddle while the agent drains it, the frog is stunned and will not move for 5 time steps. Hence, the agent's actions (in)directly affect the evolution of the stochastic environment. Walls (black in Fig. 2) or puddles cannot be traversed by frogs or the agent.

Method	Steps	Dev. π_Q	Time (ms)	MTN1
RL	215.53	—	—	1.05
UPF	244.44	21.73	36.13	0.00
PPF 65-25	216.85	2.44	11.59	0.15
PPF* 65-25	218.98	4.18	12.53	0.07
PPF 90-10	217.34	2.92	14.21	0.00
PPF* 90-10	219.67	4.84	13.42	0.00
PPF 95-5	219.03	3.11	19.56	0.00
PPF* 95-5	219.45	4.53	18.13	0.00
PPF 99-5	218.80	2.84	22.27	0.00
PPF* 99-5	220.35	4.62	24.67	0.00
PPF 99-3	218.65	2.90	40.77	0.00
PPF* 99-3	221.39	4.86	38.30	0.00

Table 1: Avg. 100 rounds, 15×15 grid, h=3, $(1-\delta)$–$\%\epsilon$ conf.–err.

RL. We trained a norm-agnostic Q-learning policy π_Q as an RL baseline. As the state space is too big for a Q-table, we encode state+action pairs using a feature vector and learn the weights accordingly. Features used are the agent's distance to the nearest active grass patch and puddle, as well as binary information about whether the agent mows a grass patch or drains a puddle in the state reached by executing a in s. The trained policy prioritizes moving towards active grass patches, but allows for small detours to drain puddles if nearby.

Norms. In our experiments we consider three distinct norms that the agent is obliged to abide by throughout the game:

- **(MTN1)** Do not capture a frog.
- **(MTN2)** Do not drain puddles where a frog is in proximity.
- **(CTD)** If a puddle with a frog in its proximity is drained, capture that frog within four time steps.

(MTN1) and (MTN2) are maintenance norms. While (MTN1) usually only requires a small detour from the optimal path, (MTN2) directly goes against the trained behavior of the RL policy. (CTD) is a contrary-to-duty norm. It activates in situations where the agent violates (MTN2) and acts as a backup to stay norm-compliant. Furthermore, (CTD) has a deadline, during which it acts as an exception to (MTN1).

For (MTN1) we track each violation, except those covered by the exception given in (CTD). For (CTD) we track all violations and additionally, how often the agent violated (MTN2), thereby obliging the agent to act according to (CTD).

Experimental Setup. The solver clingo (version 5.7.1) is used for all ASP programs, while further code and the RL implementation are in Python. The experiments were run on a Linux server with 2x Intel Xeon Silver 4314 (16 cores @ 2.40GHz, no hyperthreading) and 512GB RAM.

Experiment 1: (MTN1) Norm. In this experiment, we focus on (MTN1). By singling out one maintenance norm, we establish a fair comparison between UPF and PPF/PPF*.

Table 1 serves as a showcase of exemplary results using a horizon of 3. All policy fixing methods achieve full norm compliance, under mild configurations in line with Proposition 1. In contrast, the RL baseline on average violates the norm 1.05 times. PPF/PPF* deviate less from π_q than UPF, resulting in an average step count closer to the RL baseline and evidencing (H1). With substantially relaxed confidence

Method	Steps	Dev. π_Q	Time (ms)	MTN1	CTD [MTN2]
RL	165.55	—	—	0.65	0.52 [1.16]
UPF	199.10	22.68	675.29	0.00	0.97 [1.49]
PPF 65-25	166.68	4.83	133.98	0.01	0.02 [1.07]
PPF* 65-25	178.23	10.08	125.95	0.02	0.02 [0.96]
PPF 90-10	171.94	7.60	207.53	0.00	0.00 [1.08]
PPF* 90-10	177.89	11.61	181.76	0.00	0.00 [0.93]
PPF 95-5	172.04	8.16	292.13	0.01	0.01 [0.82]
PPF* 95-5	177.86	11.25	227.91	0.01	0.00 [0.83]
PPF 99-5	171.64	7.79	369.39	0.01	0.00 [0.97]
PPF* 99-5	179.83	11.34	279.05	0.00	0.00 [0.84]
PPF 99-3	171.42	7.80	559.82	0.02	0.01 [0.97]
PPF* 99-3	178.59	11.95	406.29	0.00	0.00 [1.00]

Table 2: Avg. 100 rounds, 50×50 grid, h=5, $(1-\delta)$–%ϵ conf.–err.

Method	Steps	Dev. π_Q	Time (ms)	MTN1	CTD [MTN2]
RL	165.90	—	—	0.80	0.32 [0.80]
UPF	195.70	21.80	3108.12	0.00	1.12 [1.40]
PPF 65-25	169.70	5.70	190.63	0.07	0.06 [0.90]
PPF* 65-25	169.00	8.00	198.36	0.00	0.00 [0.60]
PPF 90-10	167.40	4.20	295.75	0.00	0.00 [0.40]
PPF* 90-10	168.40	9.40	266.81	0.05	0.03 [0.80]
PPF 95-5	167.70	9.30	574.19	0.00	0.00 [0.70]
PPF* 95-5	175.90	15.50	489.87	0.00	0.00 [0.40]
PPF 99-5	168.40	9.10	713.78	0.00	0.00 [0.80]
PPF* 99-5	168.60	8.50	447.59	0.00	0.00 [0.70]
PPF 99-3	170.60	9.50	1201.53	0.00	0.00 [0.70]
PPF* 99-3	170.80	11.90	777.98	0.00	0.00 [1.00]

Table 3: Avg. 100 rounds, 100×100 grid, h=5, $(1-\delta)$–%ϵ conf.–err.

and error tolerance, PPF/PPF* struggle to maintain norm compliance and performance gains are insignificant (cf. setting 65-25). Results for grids 50×50, 100×100 and horizon $h = 5$ exhibit the same trends and are reported in our repository.[1]

Experiment 2: All Norms. In this experiment, we enforce all norms over a horizon of 5. We increased h from 3 to 5 as (CTD) contains a deadline of 4, hence requiring $h > 4$ for our framework to adequately plan ahead. As there is no documented method for UPF to adhere to multiple norms, we resort to only enforcing (MTN1) for that variant.

Tables 2 and 3 show exemplary results. Again, all policy fixing methods fully prevent the agent from violating norm (MTN1), under mild configurations as above. Notably, in some configurations norm violations increase with relaxed δ, ϵ, but are still within the error tolerance. Similar to Experiment 1, PPF/PPF* deviates from π_Q fewer times than UPF.

In addition to (MTN1), PPF/PPF* account for (MTN2) and (CTD). In Table 2 the RL baseline triggers (MTN2) 1.16 times on average, followed by 0.52 violations of (CTD), activated by the violation of (MTN2). As UPF always tries to avoid capturing frogs and not considers additional norms, it activates (CTD) on average 1.49 times by violating (MTN2) and subsequently violates (CTD) 0.97 times. In contrast, PPF/PPF* generally achieve full norm compliance under mild configurations, evidencing (H4). Both versions trigger (CTD) by violating (MTN2) less often than UPF. More importantly, both are able to recover from their norm-violating state, resulting in no (CTD) violations. In Table 3 the results are similar.

Experiment 3: Big Grid. In this experiment, we focus on performance and scalability in grids of sizes up to 100×100.

Table 3 again shows exemplary results. The norm compliance, discussed in Experiment 2, by PPF/PPF* is achieved with significant less compute time than UPF. This is despite UPF using the optimization techniques discussed in [Adam and Eiter, 2025] and accounting for fewer and less complex norms, serving as evidence for (H2). PPF* generally achieves even better compute times compared to PPF, scaling in impact with confidence and error tolerance and evidencing (H3).

In Table 2, UPF has an average compute time of 675ms, increasing to 3108ms in Table 3 and showing the impact of larger sized grids. In contrast, PPF/PPF* scale more favorably, with a mild configuration of 90% confidence and $\epsilon = 0.10$ going from an average of 207ms to 295ms.

7 Conclusion

As evidenced by our experiments, the probabilistic policy fixing framework we presented improves over previous, non-probabilistic attempts at emergency planning such as [Adam and Eiter, 2025], which generalized the normative supervisor in [Neufeld *et al.*, 2021]. Beyond theoretical guarantees, we observed in practice full norm compliance in our experiments.

Furthermore, a simplified encoding, thanks to optimization against concrete sampled worlds, enables us to check against more complex norms, while improving over the non-probabilistic approach in terms of average computing time. This runtime of PPF/PPF* scales with the chosen confidence and error tolerance, as well as the difficulty of finding a fully norm compliant policy. As for the latter, the heuristic of altering a policy and checking RoT repeatedly fails inevitably in norm-violating scenarios, causing PPF/PPF* to run ASP-Gen with all N instances and worst case input size of $O(N \cdot |\Pi|)$.

Finally, while our approach allows us to check for multiple, different norms at the same time, we can only assign each sampled world a value of 0 or 1 - norm compliant resp. violating. This is because we rely on confidence guarantees from sampling a specific number of world evolutions, according to the RoT and Hoeffding's inequality. As implemented in our framework, both require that values of sampled random variables are Bernoulli distributed. Therefore, norm violations are not graded in our framework. Nonetheless, we can encode a wide variety of norms including maintenance, deadline and contrary-to-duty norms as well as exceptions to those.

Outlook. In the "generate and check"-loop of our framework, we currently reground and restart both ASP programs freshly in each iteration of the loop. As the majority of information about the current state our agent is in remains static at this point, it could be viable to explore incremental ASP, for example using clingo's multi-shot solving (MSS) capabilities [Gebser *et al.*, 2019]. Furthermore, norms are currently encoded manually in our ASP framework. Extensions of ASP such as *deolingo*, based on the recently introduced *Deontic Equilibrium Logic with eXplicit negation* (DELX) [Cabalar *et al.*, 2023] and its temporal extension [Soldà *et al.*, 2025] exist, and ensure reducibility from deontic theories to standard ASP. These additions could further strengthen the proposed framework in performance and usability.

Acknowledgements

This research was funded in whole or in part by the Vienna Science and Technology Fund (WWTF) project ICT22-023 and the Austrian Science Fund (FWF) 10.55776/COE12.

References

[Adam and Eiter, 2025] Sebastian Adam and Thomas Eiter. Asp-driven emergency planning for norm violations in reinforcement learning. In *AAAI*, pages 14772–14780. AAAI Press, 2025.

[Alviano *et al.*, 2017] Mario Alviano, Francesco Calimeri, Carmine Dodaro, Davide Fuscà, Nicola Leone, Simona Perri, Francesco Ricca, Pierfrancesco Veltri, and Jessica Zangari. The ASP system DLV2. In *LPNMR*, volume 10377 of *Lecture Notes in Computer Science*, pages 215–221. Springer, 2017.

[Baral *et al.*, 2000] Chitta Baral, Vladik Kreinovich, and Raul Trejo. Computational complexity of planning and approximate planning in the presence of incompleteness. *Artif. Intell.*, 122(1-2):241–267, 2000.

[Baral *et al.*, 2009] Chitta Baral, Michael Gelfond, and J. Nelson Rushton. Probabilistic reasoning with answer sets. *Theory Pract. Log. Program.*, 9(1):57–144, 2009.

[Berner *et al.*, 2019] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

[Brewka *et al.*, 2011] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[Cabalar and Manteiga, 2025] Pedro Cabalar and Ovidio Manteiga. deolingo: Extending answer set programming with deontic reasoning. In *DEON*, pages 61–77. College Publications, 2025.

[Cabalar *et al.*, 2023] Pedro Cabalar, Agata Ciabattoni, and Leendert van der Torre. Deontic equilibrium logic with explicit negation. In *JELIA*, volume 14281 of *Lecture Notes in Computer Science*, pages 498–514. Springer, 2023.

[Cozman and Mauá, 2020] Fábio Gagliardi Cozman and Denis Deratani Mauá. The joy of probabilistic answer set programming: Semantics, complexity, expressivity, inference. *Int. J. Approx. Reason.*, 125:218–239, 2020.

[Dai *et al.*, 2025] Jin Dai, Yunfei Gao, Chao Cai, Wei Xiong, and Manjia Liu. Uav-enabled inspection system with no-fly zones: Drl-based joint mobile nest scheduling and UAV trajectory design. *IEEE Access*, 13:10844–10856, 2025.

[De Giacomo and Vardi, 2013] Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, pages 854–860. IJCAI/AAAI, 2013.

[De Giacomo *et al.*, 2019] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications. In *ICAPS*, pages 128–136. AAAI Press, 2019.

[Fichte *et al.*, 2017] Johannes Klaus Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. Answer set solving with bounded treewidth revisited. In *LPNMR*, volume 10377 of *Lecture Notes in Computer Science*, pages 132–145. Springer, 2017.

[Gabbay *et al.*, 2013] Dov M. Gabbay, John Horty, and Xavier Parent. *Handbook of Deontic Logic and Normative Systems*. College Publications, 2013.

[Gebser *et al.*, 2014] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Clingo = ASP + control: Preliminary report. *CoRR*, abs/1405.3694, 2014.

[Gebser *et al.*, 2019] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot ASP solving with clingo. *Theory Pract. Log. Program.*, 19(1):27–82, 2019.

[Governatori *et al.*, 2007] Guido Governatori, Joris Hulstijn, Régis Riveret, and Antonino Rotolo. Characterising deadlines in temporal modal defeasible logic. In *Australian Conference on Artificial Intelligence*, volume 4830 of *Lecture Notes in Computer Science*, pages 486–496. Springer, 2007.

[Governatori, 2024] Guido Governatori. An ASP implementation of defeasible deontic logic. *Künstliche Intell.*, 38(1-2):79–88, 2024.

[Gu *et al.*, 2017] Shixiang Gu, Ethan Holly, Timothy P. Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *ICRA*, pages 3389–3396. IEEE, 2017.

[Hahn *et al.*, 2025] Susana Hahn, Tomi Janhunen, Roland Kaminski, Javier Romero, Nicolas Rühling, and Torsten Schaub. Plingo: A system for probabilistic reasoning in answer set programming. *Theory Pract. Log. Program.*, 25(2):134–167, 2025.

[Hatschka *et al.*, 2023] Christian Hatschka, Agata Ciabattoni, and Thomas Eiter. Deontic paradoxes in ASP with weak constraints. In *ICLP*, volume 385 of *EPTCS*, pages 367–380, 2023.

[Hoeffding, 1963] Wassily Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[Jovanovic and Levy, 1997] B. D. Jovanovic and P. S. Levy. A Look at the Rule of Three. *The American Statistician*, 51:137–139, 1997.

[Kabir *et al.*, 2022] Mohimenul Kabir, Flavio O. Everardo, Ankit K. Shukla, Markus Hecher, Johannes Klaus Fichte, and Kuldeep S. Meel. Approxasp - a scalable approximate answer set counter. In *AAAI*, pages 5755–5764. AAAI Press, 2022.

[Könighofer *et al.*, 2025] Bettina Könighofer, Roderick Bloem, Nils Jansen, Sebastian Junges, and Stefan Pranger. Shields for safe reinforcement learning. *Commun. ACM*, 68(11):80–90, 2025.

[le Court *et al.*, 2025] Edwin Hamel-De le Court, Francesco Belardinelli, and Alexander W. Goodall. Probabilistic shielding for safe reinforcement learning. In *AAAI*, pages 16091–16099. AAAI Press, 2025.

[Lee and Wang, 2016] Joohyung Lee and Yi Wang. Weighted rules under the stable model semantics. In *KR*, pages 145–154. AAAI Press, 2016.

[Leone *et al.*, 2006] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7(3):499–562, 2006.

[Leonetti *et al.*, 2016] Matteo Leonetti, Luca Iocchi, and Peter Stone. A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. *Artif. Intell.*, 241:103–130, 2016.

[Littman *et al.*, 1998] Michael L. Littman, Judy Goldsmith, and Martin Mundhenk. The computational complexity of probabilistic planning. *J. Artif. Intell. Res.*, 9:1–36, 1998.

[Meyer *et al.*, 1994] J.-J. Ch Meyer, F. P. M. Dignum, and Roelf J. Wieringa. The paradoxes of deontic logic revisited: a computer science perspective. 1994.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015.

[Mohri *et al.*, 2012] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. Adaptive computation and machine learning. MIT Press, 2012.

[Neufeld *et al.*, 2021] Emery A. Neufeld, Ezio Bartocci, Agata Ciabattoni, and Guido Governatori. A normative supervisor for reinforcement learning agents. In *CADE*, volume 12699 of *Lecture Notes in Computer Science*, pages 565–576. Springer, 2021.

[Neufeld *et al.*, 2024] Emery A. Neufeld, Agata Ciabattoni, and Radu Florin Tulcan. Norm compliance in reinforcement learning agents via restraining bolts. In *JURIX*, volume 395 of *Frontiers in Artificial Intelligence and Applications*, pages 119–130. IOS Press, 2024.

[Nickles, 2018] Matthias Nickles. Differentiable SAT/ASP. In *PLP@ILP*, volume 2219 of *CEUR Workshop Proceedings*, pages 62–74. CEUR-WS.org, 2018.

[Prakken and Sergot, 1996] Henry Prakken and Marek J. Sergot. Contrary-to-duty obligations. *Stud Logica*, 57(1):91–115, 1996.

[Rintanen, 2004] Jussi Rintanen. Complexity of planning with partial observability. In *ICAPS*, pages 345–354. AAAI, 2004.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nat.*, 529(7587):484–489, 2016.

[Soldà *et al.*, 2025] Davide Soldà, Pedro Cabalar, Agata Ciabattoni, and Emery A. Neufeld. Tackling temporal deontic challenges with equilibrium logic. In *AAMAS*, pages 1950–1958. International Foundation for Autonomous Agents and Multiagent Systems / ACM, 2025.

[Sutton and Barto, 2018] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction, 2nd Edition*. MIT Press, 2018.

[Watkins and Dayan, 1992] Christopher J. C. H. Watkins and Peter Dayan. Technical note q-learning. *Mach. Learn.*, 8:279–292, 1992.