TU WIEN Informatics

Deontic Challenges Combined with ASP-Planning on RL-Training for the FrozenLake and Extensions

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Master programme Logic and Computation

by

BSc Adrian Schmitt
Registration Number 11809936

to the Faculty of Informatics

at the TU Wien

Advisor: O.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Eiter
Assistance: Univ.Prof.in Dr.in Agata Ciabattoni
Univ.Prof Ezio Bartocci

Vienna, May 4, 2025

Adrian Schmitt

Thomas Eiter

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Declaration of Authorship

BSc Adrian Schmitt

I hereby declare that I have written this Doctoral Thesis independently, that I have completely specified the utilized sources and resources and that I have definitely marked all parts of the work - including tables, maps and figures - which belong to other works or to the internet, literally or extracted, by referencing the source as borrowed.
I further declare that I have used generative AI tools only as an aid, and that my own intellectual and creative efforts predominate in this work. In the appendix "Overview of Generative AI Tools Used" I have listed all generative AI tools that were used in the creation of this work, and indicated where in the work they were used. If whole passages of text were used without substantial changes, I have indicated the input (prompts) I formulated and the IT application used with its product name and version number/date.

Vienna, May 4, 2025

Adrian Schmitt

Acknowledgements

Firstly, I would like to thank my advisors Profs. Thomas Eiter, Agata Ciabattoni, and Ezio Bartozzi, for their invaluable guidance and support. Their input was instrumental in shaping the structure of this work, and their suggestions opened up various lines of inquiry that significantly contributed to my academic and practical development.

I also extend my sincere thanks to the entire research team involved in Project TAIGER for their collaborative spirit, technical assistance, and constructive discussions. The numerous team exchanges helped me implement concrete techniques and address shared challenges effectively. In particular, I would like to thank Sebastian Adam, Martin Tappler, and Emeric Alexander Neufeld, whose research and methods proved especially valuable during several methods and issues of the work.

This project was funded by an external grant from WWTF project "Training and Guiding AI Agents with Ethical Rules" (TAIGER; ICT22-023), whose support made this research possible.

Abstract

This thesis investigates the integration of Reinforcement Learning (RL), Answer Set Programming (ASP), and Deontic Logic (DL) to embed normative reasoning in autonomous agents operating in non-deterministic environments. Using an extended version of OpenAI's FrozenLake environment - with dynamic obstacles, conditional tiles, and modified rewards - this study simulates real-world dilemmas where agents must balance norm adherence with reward maximization. A framework is developed that combines Q-Learning with ASP-based planning via Potassco's Telingo, incorporating deontic operators. The framework evaluates the agent's ability to handle conflicting norms, particularly contrary-to-Duty (CTD) obligations that emerge after norm violations. Experiments assess the effectiveness of different norm-reasoning strategies, comparing internal planning models with reward-shaping mechanisms. Our results show that while agents successfully learn static norms, dynamic norms and CTDs present significant challenges. Planning-based approaches accelerate learning but incomplete models struggle with dynamic aspects, whereas reward shaping enhances norm adherence but risks suboptimal policies or slow convergence. The findings suggest that a hybrid approach—integrating norm reasoning within the agent's model alongside reward shaping—offers a promising direction for future research. This work contributes to the broader discourse on embedding ethical and legal norms in RL systems, providing insights into the complexities of automated norm reasoning and laying the foundation for further exploration in this interdisciplinary field.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Reinforcement Learning (RL) is a subfield of machine learning that uses direct rewards of actions in a given state to learn an optimal behavior policy achieving the highest predicted reward across sequence of actions [SB98]. In Q-Learning, state-action values are used to evaluate actions in all states and to propose the actions for the next steps. Moreover, combining RL with planning components can accelerate the learning process and cut off futile explorations. Non-monotonic logics, such as Answer-Set-Programming (ASP) can be used to model the environment and their solvers can be incorporated into the agent. In order to find solutions (answer set), models fulfilling the rules of the program are used to define a Gelfond-Lifschitz reduct whose minimal model is the answer set [BET11]. In RL, environmental challenges are modeled and solved to maximize expected return at all costs. Nonetheless, there exist legal, ethically-sensitive, and socially acceptable operations in any real-world domain, meaning that bots acting in the actual world must adhere to a variety of different laws.To handle norm-reasoning, Deontic Logic, in particular Standard Deontic Logic (SDL), also known as Monadic Deontic Logic, offers syntax and semantics for reasoning about obligations. Specifically, there are three main operators, namely obligatory O, permitted $P = \neg O \neg$ and forbidden $F = O \neg$ [GHP+13]. However, these rules are frequently ambiguous or contradictory, so a robot must deal with quandaries while still attempting to maximize its reward. Making "just" decisions in these situations is an unsolved research issue.

The challenge still lies in articulating complex norms in a way that computers can understand, ensuring moral behavior while optimizing the benefits of an RL environment. Furthermore, this thesis is backed by the WWTF project ICT22-023 - "Training and Guiding AI Agents with Ethical Rules" (TAIGER)[1].

[1] https://www.wwtf.at/funding/programmes/ict/ICT22-023/; accessed on May 2, 2025

1.2 Problem Statement

The purpose of this work is to conduct a case study on a non-deterministic RL-environment, where the agent models its norms by approximating SDL in ASP such that its action-planning considers deontic reasoning. By utilizing paradoxes, real dilemmas are simulated, forcing the bot to decide which norms to violate and which to keep at all costs, while still aspiring maximal rewards from the environment. The contributions are briefly summarized as follows:

- Development of a unified framework that integrates components from RL, ASP, and DL.
- Modification of the FrozenLake environment to support the testing of normative constraints.
- Integration of complex normative rules into the system to evaluate their impact on agent behavior.
- Extensive experimental analysis and discussion of the resulting learned policies.

This setup allows experiments to address various research questions, among them:

- Which norms were easy / hard to be learned?
- Which norms tend to be more often violated?
- Did the agent prioritize norm-adherence over reward-maximization?
- Were the actions of the agent consistent with the norms it learned to keep?
- Did the non-determinism lead to more violations?

The results of this work highlight various aspects and challenges involved in implementing contradictory norms in an RL-framework. The learned behavior strategies will reveal processing of norm-violations in a concrete domain, offering insights in practical automated norm reasoning. Furthermore, multiple sets of deontic rules will be evaluated and compared to each another, demonstrating the impact of various morals on the RL framework. These results may be utilized in future works to offer insights into integration of norms into RL.

1.3 Outline of the Contribution

The framework will be based on OpenAI's non-deterministic reinforcement learning environment "FrozenLake"[2]. In short, this is a discrete, non-deterministic environment, in which the agent slides across a grid-lake to reach a goal while avoiding holes. Among others, the following extensions will be applied on the original version:

- adding a second bot as a dynamic obstacle; called 'traverser'
- adding a new tile 'cracked'; becomes a hole if traverser and agent occupy it

[2] https://gymnasium.farama.org/environments/toy_text/frozen_lake/; accessed on May 2, 2025

- reward-changes
- update of slippery-probabilities

Concrete levels of the environment will be tailored to the experiments. The RL component will follow an ϵ-greedy Q-learning approach, employing planning policies based on non-monotonic logic. The planning is encoded in Potassco's Telingo solver[3], which is an extension of the state of the art solver Clingo[4] to handle ASP with temporal operators such as previous state, next state, $\Box$ (always) and $\diamond$ (eventually). Notably, the model is not a direct copy of the environment. For example, the perceived expected rewards may differ from the actual ones, and the agent lacks knowledge of the traverser's movement. Deontic operators are included as predicates in Telingo, utilizing the temporal features to represenent both permanent and conditional norms. The focus will be on conflict resolutions and contrary-to-duty obligations (CTD). CTDs are norms that become a necessity after a violation of another norm [Chi63].

As an example, let 'the agent must move towards the goal tile' and 'the agent must not be on the same tile as the traverser' be norms. The conditional rule 'if the agent and traverser occupy the same tile, then the agent must move straight' is a CTD of the second norm. Depending on the concrete environment-setup, the agent may sometimes need to move across the traverser's tile to reach the goal. To resolve these conflicts, evaluation functions will be defined. However, the latter will be configured in different ways to analyze the impact of various prioritization strategies.

Due to the non-deterministic nature of the environment and the assumption of fixed transitions in planning, this framework offers various ways to push the agent into dilemmas. Consequently, violations may occur beyond the agent's control, even if it model fully adheres to the rules. The experiments generate the target policies of predefined configurations. They are evaluated using multiple criteria, including return, rule violations, learned paths, and more.

1.3.1 Organization

This work is organized into seven chapters. The next chapter provides an overview of the state-of-the-art of the three main components of the framework. Chapter 3 describes the framework in detail, while Chapters 4 and 5 present and analyze the generated data. Chapter 6 discusses related work, and finally, Chapter 7 summarizes the findings of this study.

[3] https://potassco.org/labs/telingo/; accessed on May 2, 2025
[4] https://potassco.org/clingo/; accessed on May 2, 2025

CHAPTER 2

Background

2.1 Reinforcement Learning

Machine learning is a subfield of artificial intelligence focusing on learning from data. Its algorithms are designed to improve through experience ([Mit97], Chapter 1).
Machine learning can be broadly categorized into three main subfields, distinguished by their learning strategies and training setups: *supervised learning*, *unsupervised learning*, and *Reinforcement Learning (RL)*.

Supervised learning involves using labeled training data to fit a model that approximates a complex function. This approach is commonly applied to classification and regression tasks. In classification, each input is associated with a class label, while in regression, the data is labeled with a continuous numeric output value. The goal is to approximate the underlying function by reducing the error across the data samples ([SB98], Chapter 9). However, both the training process and the selected samples must be carefully evaluated to avoid overfitting, learning unintended correlations, or introducing biases.

In *unsupervised learning*, the input data is typically unknown and not pre-analyzed. The algorithms are designed to group, reduce, or organize the data in order to uncover hidden patterns and generate insights. Common applications of unsupervised learning include clustering, Principal Component Analysis (PCA), and Self-Organizing Maps (SOM). Unlike supervised learning, there is no concrete model or function to evaluate. Instead, the data is processed and presented in a more human-interpretable form, such as patterns, groupings, and other visual representations ([Kub21], Chapter 14).

Reinforcement Learning (RL) is a subfield of machine learning focused on learning via feedback. It is characterized by autonomous exploration of an environment to discover states and feedback, with the goal of training the most efficient policy. More specifically, RL uses the direct rewards associated with actions taken in a given state learn an optimal

behavior achieving maximal (expected) cumulative rewards over a sequence of actions ([SB98], Chapter 1).

The following subsections provide an overview of the basic setting of *Reinforcement Learning (RL)*, its challenges, *Markov decision processes (MDPs)*, and relevant algorithms.

2.1.1 Goal and Elements of RL

RL is the process of training an agent within an environment. The agent observes and interacts with the environment to receive direct feedback. Typically, a training episode consists of a limited sequence of actions or a defined time window, with terminating states potentially ending the episode prematurely. The goal of RL is to optimize the agent's behavior policy, such that the cumulative reward from all actions is maximized [MKS+15].

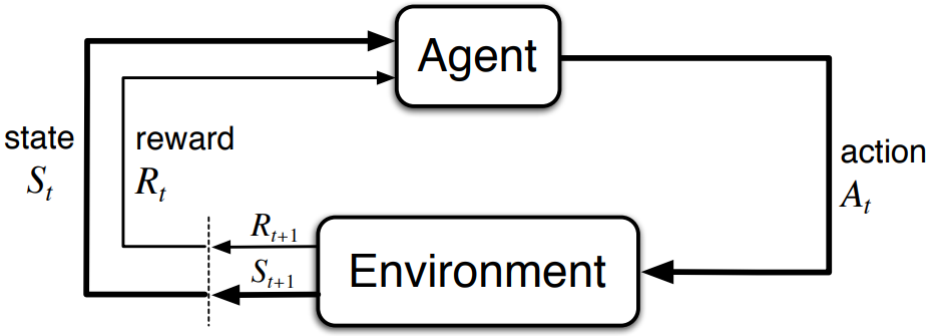

Figure 2.1: Interplay between agent and environment ([SB98], Chapter 3.1)

The core elements of RL are agents, the environment and their interactions, which are executed actions triggering state-transitions and feedback-return (see Figure 2.1). In a discrete setting, the agent selects an action A_t triggering the transition to the next state S_{t+1} and feedback of the reward R_{t+1} at time t. The reward is an objective feedback provided by the environment, with punishments being represented as negative rewards. A training episode generates a sequence of the actions, states, and rewards experienced by the agent. These sequences are used to train the agent's behavior policy. The policy is composed of learned value functions, which are typically numeric values associated with state-action pairs to quantify the expected reward. Without usage of models, the agent generally selects the action with the highest value-function in that state. An optimal policy, therefore, is a policy returning the highest amount of accumulated rewards. RL algorithms are designed to iteratively approach this optimal policy by updating the current value functions based on the rewards experienced during interaction with the environment.

Additionally, models of the environment can be used within the agent to propose subsequent actions. This enables the integration of various planning aspects and other

reasoning processes into the RL framework, a concept known as *model-based RL methods* ([SB98], Chapter 1.3).

2.1.2 Challenges of RL

An important aspect of RL, is the *exploration-exploitation-dilemma* ([SB98], Chapter 1.1). Effective exploration requires taking unknown actions and visiting new states, whereas exploitation focuses on optimizing the best-known sequences. Relying solely on exploration may prevent the agent from achieving its goal efficiently, while excessive exploitation can lead to convergence at a local optimum, missing better alternatives. Many RL-algorithms address this challenge (see Subsection 2.1.4), however, achieving a perfect balance between exploration and exploitation remains an open problem ([SB98], Chapter 2).

Many supervised learning approaches cannot be effectively applied to RL in general. In supervised learning, labeled data inputs are typically static, independent, and often easily classified by humans—making this approach particularly effective for tasks such as image recognition. In contrast, RL involves an agent interacting with a dynamic environment, where state transitions and action outcomes vary and are not associated with predefined labels or straightforward predictions. As a result, the number of learned state-action values grows exponentially, posing significant challenges ([SB98], Chapter 3). For example, training a robotic arm to stack blocks would require an extensive set of labeled training examples, which would need to be created and annotated manually. The positions of all joints can have millions of possible combinations, and the relative positions of the blocks to the robotic arm introduce further variations. In this setting, manually preparing such large amounts of input data is not feasible. Moreover, if the algorithm were tested from a different starting position, an entirely new dataset would be required. In the end, RL differs fundamentally from supervised learning and is better suited for solving interactive, goal-oriented learning problems.

Nonetheless, *advanced deep neural networks (DNNs)* can be integrated into RL, which is then called *deep reinforcement learning (DRL). Convolutional neural networks (CNNs)* and *long short-term memory networks (LSTMs)* have proven highly successful in image recognition and large language models. Their architectures enable them to process temporal contexts and handle complex, high-dimensional input structures. Incorporating these neural networks into RL enhances training efficiency in complex environments with numerous state-action pairs.
As an example, in pixel-based video games, the current screen display can serve as the state, while all possible controls represent the available actions. A DNN can process the images as input and output the expected rewards in its final layer, with each output node corresponding to a specific action. The actual rewards received from the environment are then used to train the DNN through gradient backpropagation.
Although DRL has achieved significant success in vision-based applications, it also introduces common neural network challenges into RL. These include overfitting, dependence on precise feedback, and a lack of explainability. In summary, DRL presents both chal-

lenges and valuable insights for the development of general-purpose AI systems capable of interacting with complex environments [ADBB17].

2.1.3 Markov Decision Processes

Most RL problems can be modeled as a *Markov Decision Process (MDP)* ([SB98], Chapter 3). An MDP consists of a set of states, a set of actions, a reward function, and a state-transition function. It can represent various environment characteristics, including deterministic and stochastic dynamics. This work focuses on finite, discrete, and fully observable environments. In contrast, MDPs used in partially observable environments are typically referred to as *Partially Observable Markov Decision Processes (POMDPs)*.

A MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \longrightarrow \Pi(\mathcal{S})$ is the state-transition function ($\Pi(S)$ is a probability distribution used for non-deterministic MDPs), $\mathcal{R} : \mathcal{S} \times \mathcal{A} \longrightarrow \mathbb{R}$ is the reward function and γ is the discount factor, where $\gamma \in \mathbb{R}$ and $0 \leq \gamma \leq 1$. All MDPs share the following equation for each state $S \in \mathcal{S}$ and time-step $t \in \mathbb{N}$:

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, ..., S_t] \tag{2.1}$$

This is called the *Markov Property* and means that the future is independent from the past given the present ([SB98], Chapter 3). In other words, the current state must encapsulate all relevant information from previous states and actions. Consequently, the future is determined solely by the present state. A model is considered *Markovian* if all state transitions depend only on the current state and action, without influence from past states or actions [KLM96].

Typically, in MDPs, the rewards represent the agent's objectives. By summing the rewards of subsequent states, the total gain of a given state can be computed. The *return* G_t of an episode, starting from a given state, is the discounted sum of all rewards received from the subsequent states:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.2}$$

G_t represents the total gain of all rewards following time-step t and the discount factor $0 \leq \gamma \leq 1$ weakens the relevance of future rewards ([SB98], Chapter 3.3). This means the agent prioritizes immediate rewards over future rewards, with the discount rate determining the extent of the agent's foresight. Finally, the agent has learned an optimal policy if it maximizes the return starting from its initial state.

During training, the agent needs to estimate the returns of states. By using a value function, either for states or for state-action pairs, the actual returns can be approximated. This work focuses on value functions for state-action pairs, as this is the type of value

function used in our approach (the term action-value function is also used to refer to this value function). The value function $q_\pi(s, a)$ with state s, action a and policy π is defined as follows:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \tag{2.3}$$

This equation can be simplified to the expectation of the immediate reward plus the discounted return of the next state. It can be further reduced to the sum of the next reward and a recursive call to the value function of the next state-action pair ([SB98], Chapter 3). The resulting equation is shown below:

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(s', a') | S_t = s, A_t = a] \tag{2.4}$$

Here s' and a' represent the state and action visited at time $t + 1$ under policy π, i.e. the most rewarding action in the successor state according to the agent's policy. The value-function can then be used to calculate the expected return for each state-action pair under a given behavior policy. The relationship between the value of state and the values of its successor states is called the *Bellman equation* for this value-function ([SB98], Chapter 3).

However, in most RL-problems, the goal is to find an optimal policy, rather than evaluating a given policy. Therefore, policies must be compared to each other. A policy π is superior (or greater) than policy π' if and only if the value-function $q_\pi(s, a)$ is greater than $q_{\pi'}(s, a)$, i.e.,

$$\pi \geq \pi' \leftrightarrow q_\pi(s, a) \geq q_{\pi'}(s, a) \tag{2.5}$$

for all states s and actions a ([SB98], Chapter 3.6).

The optimal policy has consequently an optimal value-function q_*, which is the maximum of all value-functions of all states and actions ([SB98], Chapter 3.6):

$$q_*(s, a) \doteq \max_\pi \; q_\pi(s, a) \tag{2.6}$$

$$q_*(s, a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\middle|\; S_t = s, A_t = a\right] \tag{2.7}$$

RL-algorithms use the *Bellman optimality equation* 2.7 to iteratively approximate q_*.

2.1.4 RL-Algorithms

Finite MDPs can be solved by three fundamental approaches, namely dynamic programming, Monte Carlo Methods and temporal-difference learning [SB98].

Dynamic Programming

First, *dynamic programming (DP)* uses the value function to structure and to search for better policies. DP applies the *Bellman equation* to iteratively update the value function, gradually approximating the optimal value function and thereby yielding the best policy. This is done by *policy iteration*, whereby each iteration consists of two main steps: *policy evaluation* and *policy improvement* ([SB98], Chapter 4).

Policy evaluation (also known as prediction) is the process of determining the value function of a policy π. The Bellman equation (see equation 2.3) is used to compute the value function for each state-action pair. Each iteration generates an updated version of the value function, until it converges.

Policy improvement (also called control), on the other hand, enhances the policy found obtained via the prediction by replacing it with a better one. By evaluating different policies, their value-functions can be compared. If the value for a given state and action is smaller than the value for the same state but with a different action, the behavior in this state can be improved. In other words, policy π can be altered into a new and greater policy π':

$$\pi'(s) \doteq \max_a q_\pi(s, a) \tag{2.8}$$

where $\max_a$ selects the a that yields the maximal value for the state s ([SB98], Chapter 4.2). This greedy policy π' cannot be worse than the original policy π by definition. In fact, the new policy will always be strictly better than the old one, unless the original policy is already optimal, in which case the new policy will remain optimal as well. Most learning methods use this greedy policy improvement and the following update rule:

$$NewEstimate = OldEstimate + StepSize \times [Target - OldEstimate] \tag{2.9}$$

The difference $Target - OldEstimate$ is the error in the current estimation. By scaling this error with a $StepSize$-parameter between 0 and 1 (also called the *learning-rate*), the update approximates the real return value ([SB98], Chapter 2.4). The $StepSize$ may be dynamic with it's own update function. However, finding the best $StepSize$ for the fastest learning on the environment is a considerable issue in RL-problems.

By combining *prediction* and *control* in each iteration, an iterative cycle of improving policies and value functions can be achieved (see Figure 2.2). The policy evaluation step requires a policy to derive a stable value function for that policy, while the policy improvement step relies on the value function to derive an improved policy. For finite MDPs, this cycle will converge to an optimal policy within a finite number of repetitions ([SB98], Chapter 4.3).

Although the time complexity is polynomial in the number of states and actions, even in the worst case ([SB98], Chapter 4.7), this approach is not practical for complex problems.

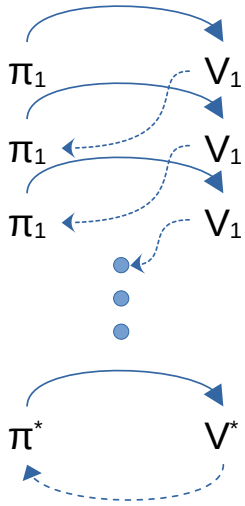

Figure 2.2: Combination of prediction and control

DP requires complete knowledge of the environment, including the full probability distribution of stochastic ones. However, since such information is often unknown or incomplete, alternative learning methods, such as *Monte Carlo* or *Temporal Difference*, are used.

Monte Carlo Methods

Monte Carlo (MC) methods learn by experience, i.e. by sampling sequences of states, actions and rewards ([SB98], Chapter 5). They are model-free and do not require complete knowledge of the environment. In contrast, *Model-based methods* can use a simulated environment to collect samples; dynamic programming is an example. Notably, policy improvement can still be applied in MC.

The *first-visit MC* estimates the value function by the average of the returns of first visits to each state s, where a first visit of the state s is the first encounter of that state in each sample ([SB98], Chapter 5.1). The value function is updated by simply adding the average difference of the returns received and the current value (see equation 2.10). Let $V(S_t) = \max_a q_\pi(S_t, A)$ for all actions A in the state S_t; then the update rule is:

$$V(S_t) \doteq V(S_t) + \frac{1}{visits(S_t)}(G_t - V(S_t)) \tag{2.10}$$

Here $visits(S_t)$ counts the first visits of this state in. The update rule for a value function for state-action pairs is similarly constructed. Since the actual return G_t is used in the update, the learning can only happen after an episode has finished.

To achieve policy improvement, MC-methods generate a sample episode, update the value functions, and improve the policy by greedily choosing the highest value in each step according to the specific update method. Additional details about MC methods are not covered here, as they are not relevant to this work.

Temporal Difference Learning

Temporal-difference learning (TD) combines MC and DP. TD updates its value function using the immediate return plus the expected return (see equation 2.11). Like MC, TD learns from samples, but it does not require a full episode to update its policy. TD uses estimates to update other estimates of the policy, similar to DP, but does not require complete models.

$$V(S_t) \doteq V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \tag{2.11}$$

The discount factor γ from above is the same discount used in the return G_t. In particular, this is referred to as $TD(0)$ or *one-step TD*, because it only uses the immediate reward of the next step. n-step TD can be expressed as $TD(n-1)$ or $TD(\lambda)$, where the next n rewards are used to update the value function. Let t be the current time-step. The difference *diff* for $TD(\lambda)$ can be approximated by:

$$diff = (\sum_{k=t}^{t+\lambda} \gamma^{k-t} R_{k+1}) + \gamma^{\lambda+1} V(S_{t+\lambda+1}) - V(S_t) \tag{2.12}$$

TD updates its estimation based on other estimations, thus TD bootstraps. While MC-methods need one full episode for each learning iteration, TD can learn incrementally during interaction with the environment, updating its estimates after each time-step. Nevertheless, both methods can lead to optimal policies ([SB98], Chapter 6.2). Generally, no method is universally superior to the other, but TD methods tend to converge faster for most practical applications. Plus, as λ approaches infinity, $TD(\lambda)$ becomes equivalent to MC.

On-policy and Off-policy

All RL-methods must address the *exploration-exploitation-dilemma* ([SB98], Chapter 5.5). In order to learn an optimal policy, non-optimal paths must be explored. However, exploration of these non-optimal paths requires a non-optimal policy. All methods discussed so far are called *on-policy-methods*, where the same policy is used both for learning optimal behavior and for interacting with the environment. So called *off-policy-methods* have two policies, namely a *target policy* used to learn optimal behavior (to satisfy exploitation) and a *behavior policy* used to investigate and interact with the environment (to satisfy exploration). On-policy methods are simpler and faster, but off-policy methods are more powerful and tend to converge slower ([SB98], Chapter 5.5).

TD and MC methods can be used as both on-policies and off-policies. There are several ways to handle off-policy methods. A common approach for the behavior policy is to introduce randomness by occasionally selecting suboptimal actions. In ϵ-greedy exploration, the behavior policy selects the greedy action with a probability of $1 - \epsilon$ and random action with a probability of ϵ-chance to choose a random action. This simple approach facilitates general exploration. Similar alterations can be used to transfer any on-policy method into an off-policy one.

Q-Learning

Q-Learning is one the most known RL-algorithms. It is typically an off-policy one-step TD control algorithm, which often uses ϵ-greedy-exploration ([SB98], Chapter 6.5). Q-Learning uses the value function of state-action-pairs:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.13}$$

In *tabular Q-learning*, the state-action values are initialized in a Q-table of size $|\mathcal{S}| \times |\mathcal{A}|$. The target policy is defined by this table, while the behavior uses ϵ-greedy selection from this table. Furthermore, Q-learning has many different variations. Most notably, *reversed Q-learning* stores visited state-action-pairs with their rewards and updates them in reversed order. This approach accelerates the back-propagation of goal rewards. While normal Q-learning can only propagate each value one step at a time in a full episode, reversed Q-learning can update the goal rewards all the way back to the starting state within the first episode. However, reversed Q-learning must wait for the episode to end before it can begin learning.

When dealing with high-dimensional sensory inputs like images, tabular Q-learning becomes infeasible due to state-space explosion and memory limitations. *Deep Q-learning networks (DQN)* address this challenge by using experience replay to store generated samples, which are then randomly grouped into mini-batches to update the neural network [MKS+13]. This relay stores a fixed amount of the most recent experiences and selects random subsets to perform an update, helping to avoid oscillations and time-dependencies in learning. Each output node corresponds to an action and the input is the full-state representation. This structure turns the DQN into a state-function that outputs values for all possible actions.

2.2 Answer Set Programming

Answer set programming (ASP) is an alteration of first-order logic used to express and to solve theories in default logic. The following subsections introduce default logic, the syntax and semantics of ASP, as well as some of its frameworks.

2.2.1 Default Logic

Default logic is a subform of *plausible reasoning*, which combines *nonmonotonic behavior* and premature inference to create new beliefs. However, the generated knowledge is considered plausible within the context the given knowledge base and it may not necessarily be correct [DM15].

Classical first-order logic is *monotonic*. For example, if A is a theory in first-order logic and w is a formula implied by A, then any superset of A also implies w ([Rei80], Section 1.2). In other words, any conclusion that can be inferred from the theory must still be derivable if the background knowledge is extended. In contrast, *nonmonotonic behavior* permits exceptions to this rule, making certain inferences impossible after learning additional information. Nonmonotonic reasoning can also be employed to fill gaps in knowledge bases by introducing consistency assumptions, such as the closed-world assumption [Rei77].

Default logic [Rei80] expresses non-monotonicity by *defaults*. Default are rules that allow "jumping" to conclusions, i.e. to create new plausible deductions. A default is a logical rule that generally holds, but may have exceptions, or it holds as long as the opposite has not been proven ([BK14], Chapter 8). Formally, a default (or default-rule) δ is defined as follows:

$$\delta = \frac{\varphi : \psi_1, ..., \psi_n}{\chi} \tag{2.14}$$

Here, φ is a first-order formula called the *prerequisite*, $\psi_1, ..., \psi_n$ are first-order formulas called the *justifications*, and χ is the resulting conclusion called the consequent ([BK14], Chapter 8.1). The horizontal line can be seen as an inference rule. A default applies, if the prerequisite evaluates to true and all justifications can be consistently assumed, meaning that $\neg\psi_i$ is not derivable under the current knowledge, for all $1 \leq i \leq n$. An applicable default is fulfilled, if the consequent holds. However, adding new knowledge can prevent the triggering of default rules and thereby allowing for the nonmonotonic behavior. A default can be simplified by omitting components, which relates to replacement with $\top$.

A *default theory* $\Delta = (D, W)$ is a pair of a set of defaults and a set of well formed formulas. A closed formula E is an *extension* of a closed default theory, if it is a fixpoint of *closure operator* Γ ($E = \Gamma(E)$) ([Rei80], Chapter 2.2), whereby $\Gamma(S)$ for any closed set S of formulas fulfills:

- $W \subseteq \Gamma(S)$

- $Cn(\Gamma(S)) = \Gamma(S)$
- If $(\varphi : \psi_1, ..., \psi_n/\chi) \in D$ and $\varphi \in \Gamma(S)$, and $\neg\psi_1, ..., \neg\psi_n \notin S$, then $\chi \in \Gamma(S)$

In other words, the extensions of default theories are deductively closed sets of formulas. These extensions contain the formulas of the theory and satisfy all the applicable defaults. An extension represents the set of conclusions that can be derived under the theory, considering the default rules that are applicable in a given context.

2.2.2 Syntax of ASP

ASP can be seen as theories within *nonmonotonic modal logics*, and normal ASP is a fragment of *Reiter's Default Logic* [BET11]. It represents form of *declarative programming* designed to solve difficult search problems and hard computational problems [Lif08]. Moreover, ASP provides an encoding of the problem itself and not a algorithm for its solution ([Lif19], Chapter 1).
ASP-programs are collections of rules consisting of a head and optionally a body. The most general rule is constructed as follows:

$$a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m,\ not\ c_1, \ldots, not\ c_n\ . \tag{2.15}$$

Whereby all letters are literals, b_i and c_j are optional and $k \geq 1$, e.g. the body can be empty, but the head has to exist [BET11]. This is a general or extended ASP-rule, however, there are more restrictive rules, like the following:

Type	Syntax	Description
fact	$a.$	has no body
Horn	$a \leftarrow \neg b_1, \ldots, \neg b_n, b_m.$	has single head-literal and body contains at most one positive literal with no default negations
basic	$a \leftarrow b_1, \ldots, b_m.$	has single head-literal and body contains no default negations
disjunctive	$a_1 \vee \cdots \vee a_k \leftarrow \ldots$	head has multiple literals connected by disjunctions
constraint	$\leftarrow b_1, \ldots, b_m,\ not\ c_1, \ldots, not\ c_n$	head is empty and body denotes states that should not hold

Table 2.1: Common classifications of ASP-rules

If not specified otherwise, extended ASP-rules (2.15) can be expected. A logic program can be classified by the most complex rule it contains, eg. a single disjunctive rule

makes the program disjunctive as well. Although defaults and non-disjunctive ASP rules correspond to one another, their semantics differ.

2.2.3 Semantics of ASP

A rule consisting only of a head is called a *fact*. For example, the rule $'employed \leftarrow .'$ means the literal *employed* evaluates to true (the arrow of a fact is usually omitted). Further, the *not* is interpreted as "not derivable" (also called default negation) and the logically not (or strong negation in ASP) is expressed by $\neg$ in front of the atoms. All c_j-literals correspond to the logically negated version of the justifications from Default Logic. An ASP-rule triggers and infers the head to the knowledge base, if the body is satisfied, i.e. if all b_i-atoms are derivable and all c_j-atoms are not derivable, then the head is inferred. As example, the program P with constant $Tweety$:

$$\begin{aligned} &bird(Tweety). \\ &canFly(Tweety) \leftarrow bird(X),\ not\ unableToFly(Tweety). \end{aligned} \tag{2.16}$$

The first rule can be read as "Tweety is a bird" and the second rule states "All birds that are not known to be unable to fly, can fly". Since $unableToFly(Tweety)$ is not derivable and $bird(Tweety)$ can be derived, the second rule can be applied to derive $canFly(Tweety)$. However, if the fact $penguin(Tweety)$ is added together with the rule that penguins cannot fly, then $canFly(Tweety)$ is no longer inferable.

Extensions also called *Answer Sets* of a program P are defined on its *ground version* $grnd(P)$ [BET11]. $grnd(P)$ is obtained by replacing all variables in the program with all possible constants from its *Herbrand Base* $HB(P)$. Formally, $HB(P)$ of logic program P is the set of all ground atoms that can be constructed from the predicate symbols of P using the constants from P's *Herbrand Universe* U. For example, the program P consist of:

$$\{\ p(1,2).\ q(x) \leftarrow p(x,y), not\ q(y).\ \} \tag{2.17}$$

Then $U = \{1,2\}$, and $HB(P) = \{p(1,1), p(1,2), p(2,1), p(2,2), q(1), q(2)\}$ and $grnd(P)$ is (by replacement of x and y):

$$\begin{aligned} \{\ &q(1) \leftarrow p(1,1), not\ q(1). \\ &q(1) \leftarrow p(1,2), not\ q(2). \\ &q(2) \leftarrow p(2,1), not\ q(1). \\ &q(2) \leftarrow p(2,2), not\ q(2).\ \} \end{aligned} \tag{2.18}$$

In order to find an answer set, models fulfilling the rules of the program are used to define a *reduct* (also known as *Gelfond-Lifschitz reduct* [GL88]) of that program under this model. If the interpretation M is the minimal model of the reduct P_M, than M is an answer set of P [BET11]. The reduct P_M is defined as the shortened version of all

rules whose default-negated atoms are not satisfied by the model M. Concretely, the reduct P_M of an interpretation M and a program P is defined as:

$$P_M = \{ \begin{array}{l} a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m \mid \\ a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m,\ not\ c_1, \ldots, not\ c_n \in P, \\ \{c_1, \ldots, c_n\} \cap M = \varnothing \quad \} \end{array} \tag{2.19}$$

To continue the example from above, an interpretation $M = \{q(2)\}$ is chosen and the reduct P_M is:

$$\{\ p(1,2).\ q(1) \leftarrow p(1,1).\ q(2) \leftarrow p(2,1).\ \} \tag{2.20}$$

However, the minimal model of this reduct is $\{p(1,2)\}$, thus the model $M = \{q(2)\}$ is not an answer set of P. Similarly, it can be shown, that the model $M = \{p(1,2), q(1)\}$ is an answer set of P [GL88]. The answer set of the *Tweety*-program (see 2.16) is $\{bird(Tweety), canFly(Tweety)\}$.

The stable model semantics can be reduced to fixpoint nonmonotonic formalisms, like default logic [GL91]. Non-disjunctive ASP-programs can be described as default theories where every justification and consequent is a literal, and every prerequisite is a conjunction of literals. If disjunctions are included then the outcomes differ; for example:

$$a \vee b. \qquad\qquad \top : / \ a \vee b \tag{2.21}$$

Both formulas represent the disjunction of a and b, on the left side as an ASP-fact and on the right as a fact in default logic. However, in ASP, there are two answer sets of $\{a\}$ and $\{b\}$, while the result of the default is the logical consequence of their disjunction, i.e. $Cn(a \vee b)$.

The solving process can be automated by using *Answer Set Solvers (ASP-solvers)* [BET11]. These solvers can be enhanced by incorporating techniques from extensively researched SAT solvers. However, most ASP solvers distinctly separate the grounding and solving phases from each other.

The solving time depends on the complexity of the ASP-program. A Horn program can be solved in linear time due to unit propagation, whereas deciding whether a non-disjunctive logic program has a stable model is NP-complete [Tho93]. Regarding disjunctive programs, checking the existence of answer sets is Σ_2^P-complete, and deciding whether a given formula holds in all answer sets is Π_2^P-complete.

2.2.4 ASP-Frameworks

Clingo[1] is developed by the Potassco (Potsdam Answer Set Solvers Collective)[2] group at the University of Potsdam. It is a widely uses tool in the ASP community, combining features from both the *Clasp*-solver[3] and the *Gringo*-grounder.

[1] `https://potassco.org/clingo/`; accessed on May 2, 2025
[2] `https://potassco.org/`; accessed on May 2, 2025
[3] `https://potassco.org/clasp/`; accessed on May 2, 2025

Clasp is an answer set solver using concepts from conflict-driven clause learning and satisfiability checking [GKNS07]. It takes a grounded logic program as input and returns satisfying answer sets. Additionally, weak constraints are used to facilitate optimization. *Gringo* is a grounder system for logic programs under answer set semantics combining approaches of *lparse* and *dlv* [GST07]. It parses an input with free variables into a finite grounded program. By using λ-restrictions the free variables in the body are bounded to remove infinite and redundant instantiations. Gringo also employs a back-jumping algorithm to efficiently handle cycles and irrelevant variables.

Clingo, however, is fusion of *Clasp* and *Gringo* to solve ASP while also offering unique controlling options [GKKS19]. Its syntax is based on ASP rules, extended with simple arithmetic operations, functions, and common aggregations.

Instead of disjunctions, Clingo uses a 'choice'-operation to handle all subsets of that choice, as an example:

$$\{p(a);\ q(b)\}. \tag{2.22}$$

The output will be four answers $\{\varnothing, \{q(b)\}, \{p(a)\}, \{p(a), q(b)\}\}$. The choice operation also allows specifying the size of the selected subsets, but its semantics still differ from that in ASP. It can be used to generate selections:

$$\begin{aligned} action(move(B1, B2)) \ :- \ & block(B1),\ block(B2),\ B1! = B2, \\ & not\ fixed(B1), not\ fixed(B2). \\ \{occurs(A, T) : action(A)\} = 1 \ :- \ & T = 0..h - 1. \end{aligned} \tag{2.23}$$

This first rule defines stacking of two blocks as a possible option, and the choice then selects an occurring action at timestamp T. The second rule combines this with weak constraints, allowing Clingo to generate solutions of combinatorial and planning problems ([Lif19], Chapter 8). There are other important ASP-solvers, like *DLV* [Nic02], but they are not considered for this work.

2.3 Norm Reasoning

Norms, rules, conventions, regulations, beliefs, and laws all share a common semantic meaning of wishful or expected behavior. *Normative reasoning* involves analyzing and evaluating actions to determine their acceptability or ethicality within a given set of norms. More concretely, a *normative system* defines behavior constraints for agents, corresponding to obligations [GMD07]. These systems are challenged by the complexity and inconsistency of real-world norms, making classical logic in general inapplicable.

2.3.1 Deontic Logic

Deontic logic is a branch of modal logic that uses specialized operators to express norms. Leibniz stated three fundamental deontic categories, namely the *obligatory* (debitum), the *permitted* (licitum), and the *prohibited* (illicitum) ([GHP+13], Part 1). Obligations O should hold in general, prohibitions F are forbidden and permissions P are allowed. These *deontic operators* are interrelated, e.g. $F\varphi = O\neg\varphi$, $P\varphi = \neg F\varphi$. Moreover, some deontic logics distinguish between strong and weak permission, whereby weak ones encompass everything that is not explicitly forbidden and strong permissions are explicitly stated in the norm system [Gov18].

However, many norms generally hold but have exceptions, and certain norms are more important than others. This results in overruling of certain laws. These aspects are handled in *defeasible deontic logic (DDL)*, an extension of defeasible logic incorporating deontic operators, which exhibits non-monotonic behavior. A basic defeasible theory is defined as a structure $(F, R, \prec)$, consisting of a set of facts, a set of rules (i.e. the norms) and a preference relation over rules [Gov18]. This system enables the derivation of definitely provable or rejectable conclusions based solely on facts and strict rules, as well as defeasibly provable or rejectable inferences, which are validated through argumentative structures. In essences, the rules for deriving a formula are compared with the counterarguments, and if the justifications for the inferences are more specific or more preferable than the ones against it, the formula is defeasibly provable. For example, the norm 'You should not enter private areas' is defeasible by another norm 'If a person calls for help from within a private area, you are allowed to enter'. The preference relation assigns priorities to laws to resolve potential conflicts.

There are various types of deontic logic, each addressing different challenges. Although they share the same deontic operators, they differ in structure, axioms, and semantics. *Dyadic deontic logic* uses conditional structures to express norms [TÅ91], e.g. $O(\psi\backslash\varphi)$ means that the 'best' φ-states must be ψ-states or in other words it's obligatory that ψ follows φ. Other examples include *Normative temporal logic* [GMD07] and *input/output logic* [MvdT00]. The former incorporates elements from *temporal logic CTL*, while the latter models a 'blackbox' reasoning system where norms are triggered by specific inputs, allowing for different operational modes.

2.3.2 Standard Deontic Logic

Historically, deontic logic has strong connections to alethic modal logic ([GHP+13], Part 1), leading early approaches to adopt the structures of necessity and possibility. *Standard deontic logic (SDL)*, also known as monadic deontic logic, is one of the earliest and most extensively studied deontic frameworks.

Syntax of SDL

The language of monadic deontic formulas ϕ is defined using the following Backus-Naur Form ([PvdT18], Chapter 1):

$$\phi = \top \mid p \mid \neg\, \phi \mid (\phi \wedge \phi) \mid O\phi \tag{2.24}$$

In here, p represents an atomic proposition. This form is syntactically complete for all classical and deontic operators in SDL. Deontic operators also have a defined scope, i.e. $O(\phi) = O\phi$, but $O\phi \wedge \varphi$ and $O(\phi \wedge \varphi)$ are distinct formulas.

Depending on the choice of axioms, there are different proof systems for SDL. System D contains the following axioms:

PL	$\vdash \phi$, ϕ is tautology
MP	If $\vdash \phi$ and $\vdash \phi \to \psi$ then $\vdash \psi$
O-K	$\vdash O(\phi \to \psi) \to (O\phi \to O\psi)$
O-D	$\vdash O\phi \to P\phi$ equivalent: $\neg O\bot$
O-Nec	If $\vdash \phi$ then $\vdash O\phi$
RMD	If $\vdash \phi \to \psi$ then $\vdash O\phi \to O\psi$

Table 2.2: Axioms of system **D** ([PvdT18], Chapter 1;[GHP+13], Part 1)

O-K and *O-D* are deontic analogues of the modal axioms *K* and *D*. System *D* is decidable, sound and complete. Furthermore, some literature refers to this system by *KD*, as it incorporates axioms from the Kripke system *K*. The distinctions between O and $\Box$ (respectively between P and $\Diamond$) are often ambiguous. For instance, the *O-Nec* axiom can be interpreted analogously to the $\Box$ -operator. Moreover, the *Andersonian-reduction* [And58] formally proves that the deontic operators can be reduced to alethic ones. Many stronger systems extend system D to include additional axioms, making them more robust and expressive, but these extensions are not relevant to this work.

Semantics of SDL

The semantics of SDL are defined using relational models, inspired by Kripke models. A relational model M is a tuple (W, R, V), where W is a non-empty set of states, $R \subseteq W \times W$ is a binary relation over W, and $V : \mathbb{P} \to 2^W$ is a labeling function assigning each atom p a set of states $V(p) \subseteq W$ ([PvdT18], Chapter 1). Intuitively this means a model is a directed graph where each note satisfies a set of atoms. If in addition R is serial, i.e., satisfies the property

$$\forall s \in W \; \exists t \in W \; : \; sRt \tag{2.25}$$

then the model itself is *serial*.

$M, s \models p$	iff $s \in V(p)$
$M, s \models \neg\phi$	iff $M, s \not\models \phi$
$M, s \models \phi \wedge \psi$	iff $M, s \models \phi$ and $M, s \models \psi$
$M, s \models O\phi$	iff $\forall t \in W(sRt \implies M, t \models \phi)$

Table 2.3: Satisfaction rules of SDL with relational semantics ([PvdT18], Chapter 1)

A state s satisfies a formula in model M under the conditions shown in Table 2.3. Notably, an obligation $O\phi$ holds if and only if all successor states of s fulfill the respective formula ϕ. The notation $s \models \phi$ is a shortcut for $M, s \models \phi$ if the model is clear from context. If all states s of M satisfy a formula ϕ, then the formula ϕ is *valid* in the model M.

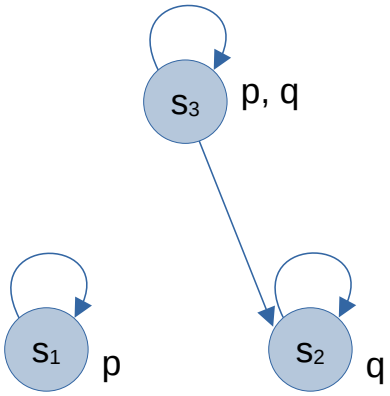

Figure 2.3: Relational model A

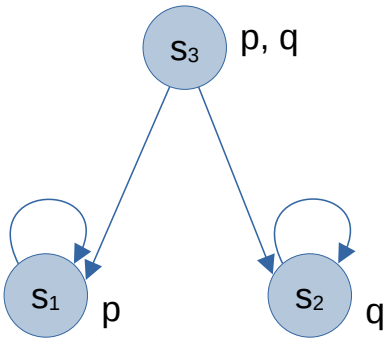

Figure 2.4: Relational model B

In Figure 2.3, there are three states with their corresponding relations and labels. The formula p holds in s_1 and s_3, but Op only holds in s_1, since s_2 is a successor of s_3 and it does not model p. However, Pp holds in both s_1 and s_3. Because the relation is reflexive, this is also an example of a serial model.

Moreover, O does not distribute over $\vee$. For example, in Figure 2.4 the formula $O(p \vee q)$ holds, since all successors model either p or q. However, $Op \vee Oq$ does not hold in s_3 and thus is a counterexample for $O(p \vee q) \rightarrow Op \vee Oq$. The natural sentence 'It's obligatory that A or B' can be interpreted in these two formulas, but they are not equivalent. This illustrates that the formulation of the problem affects the solutions. Especially the scope of the deontic operators is crucial, as example $O(p \rightarrow q)$ is different to $Op \rightarrow Oq$. There are many other aspects that may behave in counterintuitive ways or even appear paradoxical (see Subsection 2.3.3).

2.3.3 Deontic Paradoxes

Several deontic logics yield counterintuitive results because their formal properties sometimes conflict with intuitive notions of normative reasoning. This discrepancy has led to the emergence of deontic paradoxes, which have since been extensively studied. While some deontic logics can circumvent certain paradoxes, they do so at the cost of expressiveness. These paradoxes are not universally resolved but serve as sanity checks to evaluate the adequacy of a deontic logic [HCE23]. In fact, SDL is riddled with paradoxes and dilemmas.

In the following subsections, several deontic paradoxes are described, although they are often concrete examples they reveal '*wrong*' derivations of their formulations in SDL. Some paradoxes can be categorized based on their focus, for example on *RMD* (like Ross, Prior, Åqvist, Good Samaritan), on *O-K* and *O-D* (Sartre, Plato), and on deontic conditionals (Forrester, Fence, Chisholm).

The *RMD*-axiom (also called *RM*, see 2.2) enables the application of deontic operators to logical conclusions. However, when combined with classical logical inferences, it can lead to unintended obligations. Axioms *O-K* and *O-D* allow distribution of and implications among the deontic operators, but they do not provide mechanisms for resolving situations where multiple plausible obligations exist.

Deontic conditionals analyze the dependency of norms. An *according-to-duty obligation* follows a normal obligation by requiring additional imperatives. However, norms can also take the form of 'You ought to do A, but if you do not A, then you must definitely do B' [Chi63]. These are so called *contrary-to-duty obligations* (CTD). CTDs serve two primary functions: defining "acceptable" behavior in otherwise undesirable or critical situations and specifying the consequences of norm violations. Another important aspect is the type of detachment in conditional obligations of the form $p \rightarrow Oq$ and $O(p \rightarrow q)$. While an obligation triggers the other, a fact triggers the first. More concretely, if p holds, the *factual detachment* yields Oq, and if Op holds, the *deontic detachment* yields Oq. These

correspond to the narrow and wide scopes of the deontic operator O, however there is no general rule which type of detachment should be used in SDL ([GHP+13], Chapter 1).

Ultimately, deontic paradoxes highlight that while deontic logic offers a powerful framework for reasoning about obligations, it falls short of fully capturing human moral judgment. These paradoxes challenge logicians and ethicists to refine formal systems to better account for the complexities of moral and legal reasoning while recognizing that no formal system can entirely eliminate paradoxes.

Ross's Paradox

Ross's paradox reveals the differences between logical value-functions in context of imperatives. The case of disjunction is particularly interesting:

$$\begin{aligned} &A \to A \vee B \\ &OA \to O(A \vee B) \end{aligned} \tag{2.26}$$

In classical logic, $A \to A \vee B$ is valid. Applying *RMD*-axiom allows both sides to be wrapped in the O-operator. For example, if the imperative 'slip the letter into the mail-box' is satisfied, then the imperative 'either slip the letter into the mail-box or burn it' also holds [Ros44]. But burning the letter is contradicting the original obligation. Notably, the second statement holds in all models of SDL because the combination of OA and $P(\neg A \wedge \neg B)$ is unsatisfiable.

Prior's Paradox (Paradox of Derived Obligation)

For *Prior's paradox*, the material implication of classical logic was investigated in context of conditional obligations. Although original focused on conditional omniscient rules [Pri62], it reveals complications under the deontic operators.

$$\begin{aligned} &\neg A \to (A \to B) \\ &O\neg A \to O(A \to B) \end{aligned} \tag{2.27}$$

The paradox is inspired by the 'ex falso quodlibet'-principle, but deontic logic allows deriving any arbitrary obligation from a forbidden formula [Han08]. For example, from 'I shall not trespass private property, one could derive 'If I do trespass, then I must steal everything'.

Paradox of the Good Samaritan

The *paradox of the Good Samaritan* reveals issues with conjuncted statements that express good intentions but result in deriving bad obligations [Åqv67].

$$\begin{aligned} &(A \wedge B) \to B \\ &O(A \wedge B) \to OB \end{aligned} \tag{2.28}$$

If we formulate 'Smith helps Jones, who is being robbed', i.e. 'Smith helps Jones and Jones is being robbed' as $A \wedge B$, then the obligation of robbing Jones can be derived. The problem arises because the undesirable state (the robbery) is included in the good obligation (helping Jones). Similar structures in this paradox consistently lead to the undesirable state becoming a mandatory norm [Han08].

Åqvist's Paradox of Epistemic Obligation

Sometime, certain norms are only triggered if the agent is fully aware of the situation or knows necessary details. By extending SDL with a new 'knowledge'-operator K the *paradox of Epistemic Obligation* might occur. Here, $K\phi$ indicates that the agent knows ϕ, and the additional axiom $K\phi \rightarrow \phi$ holds [Åqv67]. This extended version of SDL is also susceptible to other paradoxes; for example, the Good Samaritan paradox persists even with the inclusion of the KK-operator.

$$\begin{aligned} &A \\ &OKA \rightarrow OA \end{aligned} \tag{2.29}$$

As example, let A be 'The bank is being robbed' and OKA be 'The guard is obligated to know that the bank is being robbed'. By applying the axiom $K\phi \rightarrow \phi$, OA is derivable, i.e. 'The bank must be robbed'([GHP+13], Chapter 1). This paradox demonstrates that incorporating aspects of epistemic logic into SDL generates even more paradoxes, as the combination can lead to unintended and counterintuitive conclusions.

Paradox of Free Choice

The *paradox of free choice* analyzes the implications of permission combined with disjunctions. A typical form of this paradox involves the selection of the next action, where it is permitted to choose a move from a set of possible actions.

$$\begin{aligned} &P(A \vee B) \rightarrow PA \wedge PB \\ &PA \rightarrow P(A \vee B) \rightarrow PB \end{aligned} \tag{2.30}$$

For example 'You may have cake or ice cream' implies 'You may have cake and you may have ice cream' (but not necessarily both), and this seems to hold in general for all A and B. However, from RMD we know that $A \rightarrow A \vee B$, so $PA \rightarrow P(A \vee B)$ and using the implication above this ends with PB. This means if anything is permitted, permission of any formula can be concluded. This becomes especially problematic when $B = \bot$. In this case, the formula $P\bot$ would follow, meaning that a logically impossible or contradictory action is permitted ([GHP+13], Chapter 1).

Sartre's Dilemma

Norms can be conflicting each other, but a plausible resolutions is still desirable.

$$\begin{aligned} &OA \\ &O\neg A \end{aligned} \tag{2.31}$$

Sartre's Dilemma starts with the formulation of a conflict, as example 'I must meet Mary' and 'I must not meet Mary'. By applying the *O-D*-axiom both norms become possible, although they should be excluding each other ([GHP+13], Chapter 1).

Plato's Dilemma

Often, several norms define the next expected move. While they may be logically independent and not inconsistent, they present an issue for SDL:

$$\begin{aligned} &OA \wedge OB \\ &A \wedge B \text{ is not executable simultaneously} \end{aligned} \tag{2.32}$$

In here, A and B can be replaced by 'I'm obligated to meet you now for dinner' and 'I'm obligated to rush my child now to the hospital'. Clearly, both cannot be satisfied simultaneously and resolving involves preferences of concepts of defeasibility, which SDL is lacking ([GHP+13], Chapter 1).

Forrester Paradox (The Gentle Murder Paradox)

The *Forrester paradox* analyzes CTDs and demonstrates how it is possible to conclude an obligation that contradicts the original duty [For84].

$$\begin{aligned} &O\neg A\ ;\ A \rightarrow OAB \\ &if\ A\ and\ AB \rightarrow A,\ then\ OA \end{aligned} \tag{2.33}$$

Let A be 'killing someone' and AB 'killing only gently'. The above example starts with the CTD: 'You shall not murder', but 'If you do murder, then you must do it gently'. If a murder has happened, then you are obligated to do it gently. Because a gently kill is still a kill, the *O-K*-axiom can be applied ($O(AB \rightarrow A) \rightarrow (OAB \rightarrow OA)$). After that the obligation 'You should kill someone' can be derived, and thus leading to a contradiction.

Fence Paradox

The *Fence paradox* is related to CTDs and exceptions. This paradox focuses on concrete states rather than evaluating the next actions, i.e. it deals with 'ought-to-be' instead of 'ought-to-do' [PS96]. It has several forms, but in general it reveals that obligations on abstract states always involve expectations, CTDs or other defeasible structures:

$$\begin{aligned} &\textit{there must be no fence} \\ &\textit{if there is a fence then it must be a white fence} \\ &\textit{if the cottage property includes a cliff edge, there may be a fence} \\ &\textit{there is a fence} \end{aligned} \tag{2.34}$$

The example above shows how oversimplified representations of norms can lead to contradictions in legal reasoning or ethical deliberation. Especially when the norms

are not abstract or consistent enough, the derived set of obligations differ significantly ([GHP+13], Chapter 1). For instance, questions arise such as: which rule applies for a red fence? Is it permitted to have a fence around a cottage property near a cliff edge?

Chisholm's Paradox

CTDs define norms following neglection of other duties. Chisholm analyzed the effects of CTDs in SDL and defined the *Chisholm's paradox* [Chi63]:

$$\begin{aligned}
&\textit{It ought to be that Jones goes to the assistance of his neighbors.}\\
&\textit{It ought to be that, if Jones goes to the assistance of his neighbors,}\\
&\qquad\textit{then he tells them he is coming.}\\
&\textit{If Jones does not go to the assistance of his neighbors,}\\
&\qquad\textit{then he ought not to tell them he is coming.}\\
&\textit{Jones does not go to the assistance of his neighbors.}
\end{aligned} \tag{2.35}$$

All statements are mutually consistent and logically independent from each other, but their formalization in SDL makes them mutually inconsistent or no longer independent from each other ([PvdT18], Chapter 1). Since the O-operator can be in wide or narrow scope, there are two options for the inner statements resulting in four formulations of that paradox. As an example, the following SDL formulas represent two formulations the paradox above:

$$\begin{aligned}
&Og\\
&O(g \to t)\\
&\neg g \to O\neg t\\
&\neg g
\end{aligned} \tag{2.36}$$

$$\begin{aligned}
&Og\\
&O(g \to t)\\
&O(\neg g \to \neg t)\\
&\neg g
\end{aligned} \tag{2.37}$$

Both formulations 2.36 and 2.37 express Chisholm's paradox, but they behave differently. The left formulation is inconsistent, because any model must have a state with $\neg g$, its successors must then have $\neg t$, but from Og and $O(g \to t)$, all successors must have t. Because no successor can model both t and $\neg t$, a contradiction arises. In 2.37, we have $Og \vdash O(\neg g \to \neg t)$ and thus those formulas are not independent of each other. All other formulations exhibit similar issues and in fact Chisholm's example cannot be represented in a satisfactory way in SDL ([GHP+13], Chapter 1).

2.3.4 Expressing Deontic Logic in ASP

Since both ASP and defeasible deontic logic (DDL) contain non-monotonic aspects, established ASP-solvers can substitute reasoners of deontic logic. Although it is not possible to use ASP to compute proper extensions of a DDL theory, answer sets can

still serve as a feasible solutions. As a result, DDL reasoning can be utilized as a meta-program, whereby ASP serves as the programming language for implementing DDL and its reasoning processes [Gov24].

ASP offers exception handling of weak and strong negation, allowing formulations and handling of CTDs. Defeasibility, however, requires norm-preferences to resolve situations where multiple contradicting rules may apply. In one approach, the norms are manually ordered in a superior-inferior relation and predicates for defeasible, conflict and facts are defined. This allows the ASP solver to prioritize the most superior rule in case of dilemmas and simulate defeasible extensions [Gov24].

Another approach uses ASP's weak constraints. While constraints cut the solution space, weak constraints are used for optimization. Typically, ASP solver offer weight and level for these weak constraints. By restricting their negated or violated form, norms can be formulated as weak constraints [HCE23].

Both the O-K and RMD-axioms can be simulated in ASP by these approaches. However, there is much more work done on deontic ASP, like [CCvdT23]. This and other papers are described in more detail in Chapter 6 on related work.

CHAPTER 3

Methodology

This Chapter describes the framework used for the experiments and elaborates on concrete design choices of this work.

3.1 Framework

The framework combines an RL component with an ASP planner that considers deontic norms in a specific environment called FrozenLake. It is implemented in Python and stored on GitHub [1]. Furthermore, it integrates OpenAI's Gym[2] to include the FrozenLake[3], and Potassco's Telingo[4] as the ASP solver.

The framework runs experiments based on dedicated configurations. In each experiment, a behavior policy is trained, a target policy is evaluated, and the results of both training and final evaluations are stored as outputs. Additionally, the behavior policy can utilize the ASP planner to trigger a reasoning process that optimizes the path by minimizing expected violations.

3.1.1 FrozenLake

Original Version

The *FrozenLake* is a non-deterministic environment composed of connected tiles, where each tile can be either be a hole or frozen. It also includes a starting tile and at least one goal tile. The agent attempts to cross the lake by moving to nearby tiles, but diagonal

[1] `https://github.com/Gnosling/FrozenLake_DeonticASP`; accessed on May 2, 2025
[2] `https://gymnasium.farama.org/`; accessed on May 2, 2025
[3] `https://gymnasium.farama.org/environments/toy_text/frozen_lake/`; accessed on May 2, 2025
[4] `https://potassco.org/labs/telingo/`; accessed on May 2, 2025

movements are not permitted. However, the environment may push the agent in a different way based on probability of sliding.

OpenAI is an AI research and deployment company whose OpenAI Gym (also called Gymnasium) is a toolkit for developing and comparing RL-algorithms (see [BCP+16]). It is composed of various environments ranging from Atari games and text adventures to virtual robotics and control. All environments are implemented in open-source Python code and can be adjusted as needed. Furthermore, this collection was specifically designed to support any RL research by providing predefined environments. However, we were only using the "Toy Text"-environment FrozenLake.

The FrozenLake environment is represented using ASCII characters, and its source code is available on GitHub[5]. The environment is a rectangle (or even a square) consisting of atomic tiles. Each tile is either a hole or frozen, with one frozen tile serving as the starting position and at least one other frozen tile designated as the goal. The tile types are represented by letters: S means start, G means goal, H means hole, and F means frozen. An example is shown in Figure 3.2.

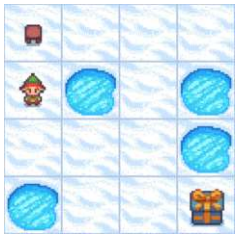

Figure 3.1:
4x4_A (pixelated)

S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

Figure 3.2:
FrozenLake-Level: 4x4_A

S	F	F	F
F	H	F	H
F	F	F	H
F	F	F	G

Figure 3.3:
FrozenLake-Level: 4x4_B

The agent begins at the starting tile and needs to reach the goal tile to successfully complete the environment while avoiding holes. The episode ends when the agent enters either a hole or the goal tile. There are four possible actions, namely left, down, right and up encoded by number zero to three in that order. All four actions are possible in each non-terminal tile. Although there is no option to wait, in some states actions can result in no movement at all. For example, if the agent is on the upper edge of the grid and attempts to move up, it remains on the same tile. The positions are represented by numbers counting the tiles, e.g. for Figure 3.2 the first row is labeled 0, 1, 2, 3, the second row 4, 5, 6, 7 and so on. Movements can then be computed by adding or subtracting either 1 or the width of the level.

The FrozenLake environment can have arbitrary configurations of holes and frozen tiles; extensions that allow for diagonal movements and/or multiple goals may be considered. However, in this work we confine to a single goal and the four original actions.

[5] `https://github.com/openai/gym?tab=readme-ov-file`; accessed on May 2, 2025

The rewards of the FrozenLake are simple. A reward of +1 is given only when the goal is reached. All other states have a reward of ±0. Notably, falling into a hole or performing a redundant action does not result in a negative reward.

This environment also exhibits non-deterministic behavior, as the lake is slippery. There is a $\frac{1}{3}$ chance that the chosen action will actually happen, while there is also a $\frac{2}{3}$ chance that the agent will slip 90° away from the intended direction. This slip can occur either 90° clockwise or 90° counterclockwise, each with a probability of $\frac{1}{3}$. Therefore, the agent can only exclude one movement direction with certainty, while all other actions have equal likelihood of occurring. Moving right, for example, has a chance of $\frac{1}{3}$ to slip right, down and up each, while sliding left is impossible. The slipperiness of the FrozenLake is unknown to the agent though.

Furthermore, there may be *safe tiles* that do not border a hole tile in their 4-neighborhood. For instance, in Figure 3.2 the first row consists only of two safe positions, namely the starting tile and the third cell. If a safe path from start to goal exist, it would be an optimal solution. Otherwise the path with the fewest unsafe tiles would yield the highest return as it has the lowest change of sliding into a hole. Figure 3.3 illustrates a FrozenLake level with no safe path, but moving along the left and lower edges forms an almost completely safe path to the goal. Additionally, by choosing to move left along the left edge, the agent can never fall into a hole, as the single adjacent hole can only be reached by sliding right, which is impossible. This strategy allows the agent to continue sliding along the left edge until it reaches the bottom-left corner, from where it can repeatedly choose to move down. Although this may take many steps, the agent will eventually reach the goal tile in a guaranteed safe manner.

Adjustments

To create more nuanced norms and fine-tune the training, several modifications were made to the FrozenLake environment. First, the slipperiness was reduced to 20%, with 10% for turning left or right each. As a result, the expected probability of executing the selected action is now 80%. This change significantly reduces the randomness in the FrozenLake, allowing the agent to have more control over its movement.

Next, a dynamic obstacle called the *traverser* was added. This 'enemy' does not harm the agent or affect the rewards. The traverser moves independently across the FrozenLake following a given deterministic pattern, which may include jumps to distant tiles. The agent, however, is unaware of the next movement of the traverser and both are allowed to occupy the same tile. Due to the hard-coded behavior, the traverser is not another agent but rather an extension to the environment itself. Moreover, a new tile was added called 'cracked' (C). This tile can only support one object, i.e. either the traverser or the agent. If both are simultaneously on the same cracked tile, then it snaps and becomes a hole, thereby terminating the episode. If there is no traverser in the level, all cracked tiles behave like frozen tiles. Furthermore, the cracks are visible to the agent.

Lastly, another tile type called 'present' (P) was added. Presents serve as an optional subgoal yielding one-time internal rewards, though the external rewards remain unchanged. In other words, the agent has an internal motivation to collect the presents, but they have no effect on crossing the FrozenLake. However, if the traverser reaches a present before the agent, it vanishes and becomes a normal frozen tile. If both the agent and the traverser land on the present tile at the same time, the agent always picks it up first. Presents cannot be placed on hole tiles, cracked tiles, or the goal, and they do not respawn. For example, Figure 3.4 contains cracked tiles and presents, that are not on the traverser's path.

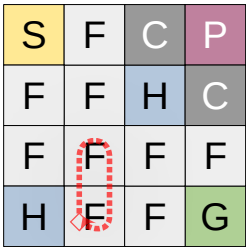

Figure 3.4: FrozenLake-Level $4x4\_C$

3.1.2 MDP-Representation

For model-free approaches the framework is reduced to RL learning. In contrast, model-based approaches can either use one model for representing the MDP and another model for planning, or a single model for both aspects. For the latter, the MDP is encoded in Potassco's *Telingo* [CKMS19]. Telingo is an extension of *Clingo* (see 3.1.4). Various strategies are available for triggering the planning component, such as random activation or activation only on first visits. While the planning component alone could solve the environment, the RL component should not be permanently overruled, as adaptive learning is necessary in dynamic environments, and exploration may lead to the discovery of more economic behaviors. Note, that the planning is also more time-consuming than the used RL-algorithms.

The MDP is composed of several Telingo files, each serving a specific purpose. One file handles the general interface of all MDPs used, while another represents the universal FrozenLake. Additional files define the specific level configuration, the deontic norms, the evaluation of violations and rewards, and the insertion of dynamic parameters. At the top, general predicates such as choosing an executable move are defined, and the Frozenlake-reasoner contains the allowed movement and expected transitions with internal rewards. Moreover, the evaluation file compares the violations and rewards to optimize the action-path. By separating these layers into their own files, the framework can use the same ASP process for various instances.

The transition itself is an ASP-solving process, whereby current states are inputted and an optimal path is computed whose first action is returned. Furthermore, a planning-horizon

Algorithm 3.1: Q-Learning

Input: learning-rate α $(0 \leq \alpha \leq 1)$, discount-factor γ $(0 \leq \gamma \leq 1)$, step-limit sl $(1 \leq sl)$, starting-state $initial\_state$, markov decision process mdp

Output: Estimation of $\pi \approx \pi_*$

$\pi \leftarrow$ initialisation dictionary;
$Q(s, a) \leftarrow$ initialisation dictionary;
$S_t \leftarrow initial\_state$;
$mdp.set\_state(S_t)$;
while $sl > 0$ *and not* $terminal(S_t)$ **do**
  $sl \leftarrow sl - 1$;
  $A_t \leftarrow \pi(S_t)$;
  $S_{t+1},\ R_t \leftarrow mdp.transition(A_t)$;
  $G_t \leftarrow R_t + \gamma \arg\max_a Q(S_{t+1}, a)$;
  $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t))$;
  $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$;
  $S_t \leftarrow S_{t+1}$;
end
return π;

parameter is used to plan up to the next n steps allowing configurations for long-term and for short-sighted planning.

3.1.3 RL-Learning

The framework uses tabular Q-learning with full state-action pairs. Depending on the initialization strategy, all values are filled with a default value, e.g. 0 or random. Q-learning is an off-policy, one-step TD control algorithm (see Chapter 2.1.4). A value function $Q(s, a)$ is used to estimate the return of taking action a in state s. The update is done iteratively, using the intermediate reward and the estimation for the next state (see Equation 2.13 and Algorithm 3.1). This algorithm uses a constant learning rate to scale the update of the value function, a discount-factor to weaken the influence of future rewards, a MDP to perform a transition, e.g. to interact with the environment, and also a step-limit-parameter used to define maximum length of a training-episode.

However, Algorithm 3.1 shows only one policy (mainly for simplification purposes). In standard Q-learning, there is typically a target policy used for training and a behavior policy used for exploration, whereby $\pi(S_t)$ defines the next proposed action. In our work, we also used both policies with different strategies. The target is a simple greedy policy always returning the best learned action, while the behavior is an ϵ-greedy policy with a chance of ϵ to execute a random action.

Furthermore, we also used reversed Q-learning (see Algorithm 3.2). Reversed Q-learning stores each visited state-action-pair, its return, and its successor-state, allowing the value

Algorithm 3.2: reversed Q-Learning

Input: learning-rate α ($0 \leq \alpha \leq 1$), discount-factor γ ($0 \leq \gamma \leq 1$), step-limit sl ($1 \leq sl$), starting-state *initial_state*, markov decision process *mdp*

Output: Estimation of $\pi \approx \pi_*$

$\pi \leftarrow$ initialisation dictionary;
$Q(s, a) \leftarrow$ initialisation dictionary;
$S_t \leftarrow initial\_state$;
$mdp.set\_state(S_t)$;
$memory \leftarrow$ empty list;
while $sl > 0$ *and not* $terminal(S_t)$ **do**
    $sl \leftarrow sl - 1$;
    $A_t \leftarrow \pi(S_t)$;
    $S_{t+1},\ R_t \leftarrow mdp.transition(A_t)$;
    $memory.append(S_t, A_t, R_t, S_{t+1})$;
    $S_t \leftarrow S_{t+1}$;
end
for S_t, A_t, R_t, S_{t+1} *in* $reversed(memory)$ **do**
    $G_t \leftarrow R_t + \gamma \arg\max_a Q(S_{t+1}, a)$;
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(G_t - Q(S_t, A_t))$;
    $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$;
end
return π;

function to be updated in reverse at the end of an episode. Because the FrozenLake only has a single feedback when reaching the goal-tile, this method enables a faster backpropagation of the final reward. For instance, in normal Q-learning, the goal reward can only be updated to its predecessor in one episode, whereas in reversed Q-learning, this reward can be propagated all the way back to the initial state in a single episode (although the reward will shrink due to the discount factor and learning rate).

The possible actions are the four movement options already described. However, the states must include more than just the agent's current position due to the dynamic aspects of the environment. In our framework, a state is represented as a tuple composed of the agent's current position, the traverser's current position, and a sorted list of the remaining presents. The Q-table stores values for all actions in each state. With n being the number of tiles, the state space consists of n options for the position of the agent, n options for the position of the traverser, and 2^n options for the locations of presents (as they are a subset of the tiles). However, by limiting the maximum of presents to three and because the majority of experiments do not include presents, the state space remains polynomial, i.e. in $O(n^2 \cdot c)$ with $c < 8$ being a constant.

Lastly the Q-Table can be initialized according to different strategies. In our experiments, we used *all_zero*, *random*, *distance*, $safe$ and so-called *potential functions* as initial-

ization methods. The *distance* strategy sets state-action pairs to the value of difference of distances of current state and successor state. $safe$ initialization gives benefits to actions leading into safe tiles (i.e. tiles with no hole next to it). The *potential functions* are defined in Section 3.1.5 and can be applied to the state-action pairs to retrieve values that represent their norm compliance.

3.1.4 ASP-Planning

This component uses ASP to model the environment and plan the next actions. It searches for an optimal path by maximizing internal rewards, minimizing violations of norms and by comparing and resolving conflicting norms (see Chapter 3.1.5). The planner outputs the first action of the optimal path and is triggered according to a given planning strategy. It plans up to the next n actions to yield the expected 'best' path, whereby n is the *planning_horizon* parameter defined by the experiment configuration. Ideally, this planning should provide the agent with a start bonus and positively influence the learning speed. However, the planning component only serves to boost the RL learning, and it is not used to solve the MDP though.

The integration of the planning-model is similar to the MDP model as discussed earlier, consisting of models for the abstract MDP, the general environment, the concrete level-data, the deontic reasoning, etc. However, the environment and the planned model vary from each other, such as by having different rewards, restricting certain moves and by employing slipperiness. By using an internal reward of -1 for any action, the agent is penalized for taking redundant steps. Dynamic aspects are inputted in the model at each step, as example, the current position of the agent, of the traverser, and the remaining presents are stored in a temporary file. While the observable layout is fully represented, the agent remains unaware of the sliding probabilities and the traverser's path. Thus, the model is incomplete.

Telingo[6] is an extension of *Clingo* with temporal operators over finite linear time, that uses multi-shot solving for unfolding solutions over sequential states [CKMS19]. It allows for planning in dynamic situations with temporal conditions. Moreover, formulas can be grouped into active sections:

#program always.	rules apply in all states
#program initial.	rules apply in first state
#program final.	rules apply in last state
#program dynamic.	rules apply everywhere, except first state

Table 3.1: Program parts of Telingo

[6] `https://potassco.org/labs/telingo/`; accessed on May 2, 2025

Telingo supports typical boolean operators and advanced time-constructs and bounded operators, like dynamic ranges, until ϕ, eventually, etc. It can also refer to successor and predecessor states of predicates. For instance, $'p(X)$ is the value of p in the last state and $p'(X)$ is used to refer to values of the next state. In this work, Telingo is primarily used to express the state transitions and previous moves within the FrozenLake.

The general-reasoning file selects the next action (see Listing 3.1).

```
#program always.
{act(move(X)) : executable(move(X)), allowedAction(move(X))} = 1 :-
    not terminalStateReached.
```

Listing 3.1: Excerpt from general_reasoning.dl

Dynamic parameters are inputted as facts into a special file, as example:

```
#program initial.
currentState(4).
currentStateOfTraverser(7).
lastPerformedAction(move(down)).
```

Listing 3.2: Excerpt from dynamic_parameters.dl

The actual planning model therefore only needs to represent the environment. See Listing 3.3:

```
#program always.

% define possible actions
executable(move(left)) :- currentState(L), frozen(L-1), not leftEdge(
    L), not terminalStateReached.
executable(move(down)) :- currentState(L), frozen(L+width), not
    lowerEdge(L), not terminalStateReached.
executable(move(right)) :- currentState(L), frozen(L+1), not
    rightEdge(L), not terminalStateReached.
executable(move(up)) :- currentState(L), frozen(L-width), not
    upperEdge(L), not terminalStateReached.

% assume traverser stays
currentStateOfTraverser(X) :- 'currentStateOfTraverser(X).

% define transitions
currentState(L-1) :- 'act(move(left)), 'currentState(L), not leftEdge
    (L).
currentState(L) :- 'act(move(left)), 'currentState(L), leftEdge(L).
currentState(L+1) :- 'act(move(right)), 'currentState(L), not
    rightEdge(L).
```

```
16 currentState(L) :- 'act(move(right)), 'currentState(L), rightEdge(L).
17 currentState(L-width):- 'act(move(up)), 'currentState(L), not
       upperEdge(L).
18 currentState(L):- 'act(move(up)), 'currentState(L), upperEdge(L).
19 currentState(L+width) :- 'act(move(down)), 'currentState(L), not
       lowerEdge(L).
20 currentState(L) :- 'act(move(down)), 'currentState(L), lowerEdge(L).
21 :- currentState(X), currentState(Y), X != Y.
22
23 % define termination
24 goalStateReached :- currentState(S), goal(S).
25 failedStateReached :- currentState(S), hole(S).
26 failedStateReached :- currentState(S), currentStateOfTraverser(S),
       cracked(S).
27 terminalStateReached :- goalStateReached.
28 terminalStateReached :- failedStateReached.
```

Listing 3.3: Excerpt from frozenlake_reasoning.lp

As already mentioned, the states of FrozenLake are represented by the current position, and the tiles are expressed as positive integers in a grid-array (see Subsection 3.1.1). Thus, state-transitions can be computed. For example moving left corresponds to subtracting 1 from the current position. Additionally, specific data such as the goal, current position, holes, and edges are added dynamically. Because all rules are within # *program always*, they apply to every state.

Furthermore, this model prevents movement against edges, e.g. moving up is prohibited, if the agent is on the upper edge of the FrozenLake. It also does not model the slipperiness of the lake, as all actions are modeled fully deterministic. While the FrozenLake only has a reward of $+1$ on the goal-tile, this model has a negative reward of -1 for each move, -100 for falling into a hole, $+100$ for reaching the goal, and $+30$ for picking up a present. These rewards are defined in dedicated files. However, the rewards in the planning model are not utilized in the RL component, as the primary objective remains successfully crossing the lake. Additionally, the agent assumes that the traverser remains on its current tile, as it cannot reliably predict its movement.

Next, the planning-strategies define the triggering of the planner. When activated, the planner's output overrides the proposal from the Q-table. There are five strategies, namely:

- *no_planning* (model-free)
- *full_planning* (triggers always)
- *plan_for_new_states* (only plans on first visits of states)
- *delta_greedy_planning*
- *delta_decaying_planning*

The parameter *delta* is specified by the configuration and is used to compute the probability of triggering. Specifically, δ represents the probability in the greedy strategy, while

the decaying strategy uses the function $decay\_value(n) = e^{-\delta \cdot n}$, where the decay starts initially with value 1 and n is the number of previous steps.

Lastly, the planner operates as a dedicated subprocess that generates all shortest paths for each step-size up to the *planning_horizon* within a time limit of 60 seconds. During postprocessing, the optimal action for each solution length is extracted from the output and returned to the policy class.

3.1.5 Deontic-Integration

The deontic aspects are handled by Telingo as additional inputs. The norms are organized into distinct files, and the evaluation strategies are also stored separately.

There are 10 norms being used in the experiments and they are tracked as their negation, i.e. their violation (see Table 3.2).

notReachedGoal	the agent did not end the episode on the goal tile
occupiedTraverserTile	the agent moved on the current tile of the traverser
turnedOnTraverserTile	the agent was on the traverser-tile and did not repeat its last performed action
stolePresent	the agent entered a tile with a present
missedPresents	the agent did not collect all remaining presents before the episode finished
movedAwayFromGoal	the agent moved to a tile strictly further away from the goal
didNotMoveTowardsGoal	the agent did not move to a tile strictly closer to the goal
leftSafeArea	the agent moved from a safe tile to an unsafe one
didNotReturnToSafeArea	the agent was on an unsafe tile and did not move on top of a safe one

Table 3.2: Norms as violations

notReachedGoal is a simple 'success'-norm. Because the FrozenLake has a single reward of +1 at the goal, this norm corresponds to the return of the episode. The traverser is a dynamic obstacle affecting the norms **occupiedTraverserTile** and **turnedOnTraverserTile**, where the latter is a CTD of the first one. The norms related to presents are in conflict with each other and will influence the internal rewards of the

planning component. Specifically, the agent must evaluate whether the benefit of picking up a present outweighs the violation of stealing it. Furthermore, there are two norms affecting progress towards the goal. Both **movedAwayFromGoal** and **didNotMoveTowardsGoal** are rules to get closer to the goal, however the first one permits no real movements (like sliding into the same position on edges), whereas the second one does not. Finally, some tiles can be considered safe, where a tile is *safe* if it and its four adjacent neighbors are not hole tiles. If the agent moves only on safe tiles, then all holes are completely avoided. **leftSafeArea** instructs the agent to remain on safe tiles and **didNotReturnToSafeArea** is thus a CTD.

The set of norms are stored in separate files. Each norm is represented as a deontic operator containing a derivable atom. If the atom is inferred, then it is added as a $currentViolation(\dots)$ into the respective step of Telingo. The CTDs use information about the last performed action, which is inserted dynamically. Note that the performed action refers to the movement executed, which may differ from the intended one. Additionally norms are assigned a level, which is used in the evaluation to prioritize and weight rules. The higher the level, the more important the norm is. An example is shown in Listing 3.4, which defines **occupiedTraverserTile**, **turnedOnTraverserTile** and **notReachedGoal**.

```
1  #program always.
2
3  % The agent must not be on the same tile as the traverser
4  forbidden(occupiedTraverserTile).
5  occupiedTraverserTile :- currentState(X), currentStateOfTraverser(Y),
       X=Y.
6  currentViolation(forbidden(occupiedTraverserTile)) :- forbidden(
       occupiedTraverserTile), occupiedTraverserTile.
7  level(occupiedTraverserTile,2).
8
9  % If the agent and traverser occupy the same tile, then the agent
       must move straight
10 forbidden(turnedOnTraverserTile).
11 turnedOnTraverserTile :- occupiedTraverserTile, act(move(X)),
       lastPerformedAction(move(Y)), X != Y.
12 currentViolation(forbidden(turnedOnTraverserTile)) :- forbidden(
       turnedOnTraverserTile), turnedOnTraverserTile.
13 level(turnedOnTraverserTile,3).
14
15
16 % The agent must reach the goal tile
17 obligatory(reachedGoal).
18 level(reachedGoal,4).
19 #program final.
20 currentViolation(obligatory(reachedGoal)) :- not reachedGoal.
```

Listing 3.4: Excerpt from deontic_norms_2.lp

The active norms and their level are defined by the respective file stated in the experiment configuration. All norms with their default settings can be found in the Appendix (see Listing 7.1).

The model is designed to find an optimal path for solving the level, which involves simultaneously maximizing rewards and minimizing violations. To configure this multi-objective optimization, different evaluation strategies are employed. The most commonly used setting applies weak constraints to prioritize by weight (see Listing 3.5). Other evaluation strategies are also available, including optimizations focused on either rewards or norms, or a scaling function that quantifies both rewards and violations into a single objective (see Listing 3.6).

```
1 benefits(E) :- rewards(R), E = 1 * R.
2
3 #program final.
4 #maximize {E@1 : benefits(E)}.
5 #minimize {V@L : violationsOfLevel(V,L)}.
```

Listing 3.5: Excerpt from evaluations_4.lp

```
1 penaltyPerLevel(X,L) :- X = #sum { V * U : violationsOfLevel(V,U), U=
     L}, violationsOfLevel(_,L).
2 penalty(P) :- P = #sum { X, L : penaltyPerLevel(X,L) }.
3 benefits(G) :- rewards(R), G = 1 * R.
4
5 eval(E) :- penalty(P), benefits(G), E = G - P.
6
7 #program final.
8 #maximize {E@1 : eval(E)}.
```

Listing 3.6: Excerpt from evaluations_2.lp

Telingo optimizes the objective with the highest level first for each step size, regardless of the lower-level objectives. Each violation is counted at most once per step. If a norm has a level of 1, its violation will be evaluated against a reward of 1, meaning the violation results in a penalty of -1 into the benefits. Furthermore, other evaluation strategies focus exclusively on rewards, only violations, or use an objective function that translates both violations and rewards into a numeric value.

The output of the planning component is an optimal path for each step size up to the value of *planning_horizon*. A post-processor is applied to identify the best path size and extract its corresponding action as the agent's selection. The optimal path is the shortest path with the highest value based on the highest optimization criteria. In addition, a cache is implemented to store past computations. If the inputs are identical, the cache directly outputs the planned action without triggering the actual solving process.

3.1.6 Norm-Enforcing

The deontic integration enables the agent to use planning to follow norms during the training phase, with the expectation that it will learn lawful behavior in the final Q-table. However, additional enforcements can be employed to further improve the agent's behavior. These deontic reinforcers can be applied at different stages of the RL cycle (see Figure 3.5).

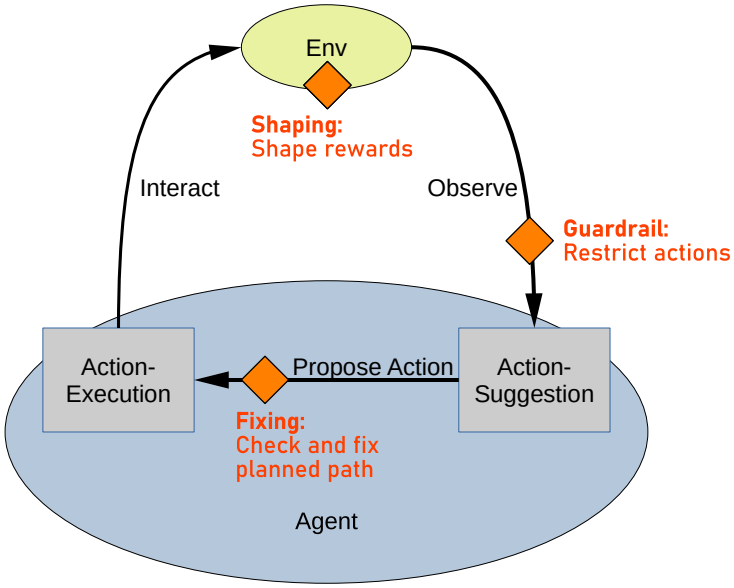

Figure 3.5: RL-cycle with markings for enforcements

In this work, three enforcement strategies were tested, namely guardrailing of allowed actions, fixing of proposed actions and shaping of rewards. These strategies can be combined and activated both during and after training. During training, the enforcement is applied on the behavior policy, whereas after training, the target policy is evaluated first without and then with enforcement. Moreover, the enforced norms may differ from the ones used in learning. Depending on the strategy, enforcing the most critical norms or all norms will yield different results.

Action-Guardrailing is a restriction of the allowed actions before they are passed to the agent. The expected next state for each action is constructed, the violations are computed, and then a set of actions with minimal violations is returned. The component does not use planning and checks only the next step. However, these restrictions could potentially cut off optimal paths and limit explorations.

Next, ***Policy-Fixing*** triggers the planning component after the agent made its suggestion for the next action. It constructs a suggested path of a given length from the state-action

pairs in the Q-Table. For instance, the first action, 'down', has an expected successor state, and the highest action value in that state determines the next action. The fixing then uses planning to validate this path and in case of violations, a fix is planned, with the first action of the fixed path executed instead of the original suggestion. Unlike the normal planning, this approach can enforce different norms and also use separate horizons for checking and fixing. As example, it may check the next three steps and construct fixed paths of greater length. Nonetheless this component overrules the policy and should therefore only be used for the most critical steps. Otherwise it would replace the full reinforcement learning process, rendering the Q-Table obsolete.

Lastly, ***Reward-Shaping*** directly changes the rewards from the environment to incorporate feedback of norms. The agent itself is not changed or enforced, instead an optimal policy should be learned that will remain optimal in the original environment. However, not all environment-changes are policy-invariant, and thus rewards cannot be altered arbitrarily. To avoid learning sub-optimal policies, a *potential-based reward shaping function* can be used to modify the rewards [NHR99]. A function F is potential-based if its defined by the difference of potential values between states. This function F can be added to the original rewards to get the changed rewards R':

$$\begin{aligned} R'(s,a,s') &= R(s,a,s') + F(s,a,s') \\ F(s,a,s') &= \gamma\Phi(s') - \Phi(s) \end{aligned} \tag{3.1}$$

In 3.1 rewards are defined based on the current state s, the executed action a, and the successor state s'. Additionally, Φ represents the potential function of states, which is used to determine the lawfulness of all states. The less violations occur in a state, the higher its value. Since the original rewards are $+0$ for almost all states, the shaped rewards will essentially define the learned policy. A state on the Frozenlake is represented as a tuple (p_A, p_T, p_P) of the agent's position p_A, the traverser's position p_T and the positions p_P of presents. Thus this state-function can be defined for the simple norms as in Listing 3.7:

```
1     elif norm == "missedPresents":
2         if terminated and len(state[2]) > 0:
3             value += -len(state[2]) * (scaling_factor**(
                  level_of_norms[norm]-1))
4
5     elif norm == "movedAwayFromGoal" or norm == "
          didNotMoveTowardsGoal":
6         value += (-distance_to_goal(state[0], goal, width, height) /
              0.8) * (scaling_factor ** (level_of_norms[norm] - 1))
7
8     elif norm == "leftSafeArea":
9         if _tile_is_safe(state[0], env):
10            value += 1 * (scaling_factor**(level_of_norms[norm]-1))
```

Listing 3.7: Excerpt from `get_state_value(···)`

The value is increased with each fulfilled norm and the increment is scaled exponentially with the level of the norm. Thus, Φ returns a positive numeric value for the norm adherence. Furthermore, the difference between the successor-value and the state-value guides the agent towards better states, while also preventing positive feedback loops. Despite that, the CTDs cannot be evaluated by considering just a single state, as they need information from previous states and actions. For example, the violation **didNotReturnToSafeArea** is triggered by two consecutive states that are both not safe.

Although the potential-based function guarantees the optimality of the learned policy, this approach is unsuitable for expressing the CTDs. However, by using a state-action penalty the rewards can be shaped to punish violations:

$$\begin{aligned} R'(s,a,s') &= R(s,a,s') + F(t_n) \\ F(t_n) &= \gamma\Phi'(t_n) - \Phi'(t_{n-1}) \end{aligned} \tag{3.2}$$

In 3.2, Φ' takes as input the trail t_n of previous states and actions, with s' being the latest state. The function evaluates the full history of the trail and checks for violations. Instead of positive values, penalties are computed and summed up, and the norms are scaled according to their level, similar to the state-function. Nonetheless, there is no guarantee that rewards shaped by Φ' will yield an optimal policy of the original environment.

Although Φ adds positive rewards and Φ' negative ones, both functions punish moving to worse states. In fact, Φ' is never going to return a positive reward, since each t_n contains at least all violations from t_{n-1}. Moreover, the discount factor $\gamma < 1$ gives a slight punishment, even if the violations in inputs are the same. This essentially penalizes every action by default.

```
1    norms = _check_violations(norms, trail, terminated, env)
2
3    penalty = 0
4    for norm, violations in norms.items():
5        if violations > 0:
6            penalty += violations * (scaling_factor**(level_of_norms[
                 norm]-1))
7    penalty = -penalty /100  # Note: this is to scale all rewards
         down, penalties are negative
```

Listing 3.8: Excerpt from `get_state_action_penalty(···)`

Furthermore, both shaping functions can be used as initialization strategies for the Q-Table. That means either $\Phi(s)$ or $\Phi'(\{s,a,s'\})$ is added to $Q(s,a)$, where s' is the expected successor. However, the norm **turnedOnTraverserTile** cannot be considered in the initialization of state-action pairs, since two consecutive actions need to be checked. The CTD **didNotReturnToSafeArea**, on the other hand, can be checked with s and its expected successor though.

For discrete-states environments, rewards shaping in the environment and during initialization are equivalent, if potential-based functions are used [Wie03]. The function for state-actions penalties can be expected to perform similar between shaping and initialization as well. By using this approach, the expected best path is implemented in the Q-Table before any learning is applied. Although the RL-cycle is not directly affected by this, it can be assumed to significantly speed up the learning process.

3.2 Design Choices

3.2.1 Non-deterministic Environment and Deterministic Planning

It might seem unusual to use a deterministic planner for a non-deterministic environment, but we intentionally employed deterministic planning to analyze the agent's learned behavior. However, adjusting the slipperiness of the FrozenLake was necessary to ensure this planning is suitable. While the original chance of $\frac{1}{3}$ for moving straight, sliding left, or sliding right respectively is too unpredictable to apply a deterministic planner, the adjusted sliding-probabilities are relatively compatible with the planning. Moreover, the model is not supposed to solve the environment in the first place, instead it should accelerate learning and offer deontic reasoning. Because the slipperiness and optimal paths are learned though RL, the agent does not require a stochastic planner to solve the FrozenLake.

Although the agent recognized all positions, the environment is effectively partly observable. Additionally to the real distribution of transitions, the movement of the traverser is unknown to the model and is assumed to be stationary. This setup challenges the agent with varying movement patterns of the traverser, requiring it to potentially detour from a previously planned path. Despite the possibility of including a trail of the traverser into the model to predict its next position, this was not implemented intentionally. The goal was to observe what happens when the agent must reliably learn norms that are influenced by other actors who are 'unpredictable'.

3.2.2 Integration of RL, ASP and SDL

We chose a simple environment with an obvious success path to analyze the impact on the learning if norms were added. The raw planning is a speed up of the RL-Learning and yields a model in which norms can be represented. Because ASP has connections to SDL and since it can effectively be used for problem solving and modeling, it was selected as the planning formalism. The RL cycle and the discrete environment are also compatible with RL enhanced by an ASP-model. Initially, we did not modify the environment to include norms. The norms are part of the internal model, allowing for testing of agents with different moralities on the same MDP. However, integrating norms directly into the rewards is an alternative approach worth analyzing (i.e. rewards shaping). If the

environment itself contains all the norms in its feedback or possible actions, then the optimal solution would already be lawful.

The norms could potentially conflict with the original rewards. The planning component needs to quantify and to evaluate not only rewards against violations, but also the norms amongst themselves. Because there is no universal solution, different evaluation strategies were developed. The norms are grouped into distinct sets to exhibit how dedicated combinations may affect the learning. These constellations are inspired by some of the deontic paradoxes (see Section 2.3.3). The defined norms include CTDs and contradictions, which will be used in the main experiments. While many classical deontic paradoxes are formulated in natural language and rely on implicit uncertainty or context, ASP is based on the non-monotonic behavior and the closed world assumption. This makes direct reproduction of some paradoxes difficult or even impossible in ASP. Rather than capturing their full natural-language ambiguity, ASP often resolves many paradoxes deterministically due to its formal semantics and avoids common pitfalls of SDL. However, there is a part of the experiments that focuses on exploring different representations of the deontic aspects and simulations of expected behaviors under paradoxical norms in ASP within the context of the FrozenLake.

CHAPTER 4

Experiments

This Chapter provides an overview of the experiments and generated charts, outlines the experimental hypotheses, presents the results, and includes a brief summary for each experiment suite.

4.1 Overview

The experiments are defined by configurations. An example of such a configuration is shown in Listing 4.1.

```
"A1": {"repetitions": 100, "episodes": episodes_3x3,
    "max_steps": max_steps_3x3, "evaluation_repetitions": 100,

    "frozenlake": {"name": "FrozenLake3x3_A", "traverser_path": "3
       x3_A", "slippery": True},

    "learning": {"norm_set": None, "epsilon": epsilon, "
       initialization": "distance", "reversed_q_learning": True, "
       discount": 0.99, "learning_rate": 0.3, "learning_rate_strategy
       ": "constant", "learning_decay_rate": None},

    "planning": {"norm_set": 1, "delta": None, "strategy": "
       no_planning", "planning_horizon": None, "reward_set": 2},

    "deontic": {"norm_set": 0, "evaluation_function": None},


    "enforcing": None,
}
```

Listing 4.1: Configuration: A1

The parameters in Listing 4.1 are grouped into blocks. The first one contains general data, like number of repetitions of the experiment, number of episodes and number of evaluation repetitions, which are the amount of tests done for each final target episode, such that the average values can be evaluated. The FrozenLake is defined by the level, slipperiness, and the movement pattern of the traverser (which is for our experiments always a path). The learning-section contains typical RL-parameters, such as discount and epsilon, as well as specific configurations, e.g. the strategies for the learning rate and initialization. The parameters for the planning and deontic sections are described in Subsections 3.1.4 and 3.1.5 respectively. Different norm sets are configured for different purposes within these blocks. While the norms in the learning-block are only used if the potential function is chosen as the initialization strategy, the norms in planning for the planning component, and the norms in the deontic-block are used the evaluation. If not specified otherwise, all norms are considered for evaluation.

Certain parameters are fixed across all experiments, while others vary based on level sizes. In general the most important parameters are the norm sets and the strategies. All configurations can be found in $src/configs.py$, all results files and all plots are also on Github[1]. In addition, all level structures are contained in the Appendix (see Chapter 7).

There are five suites of experiments, labeled $A - E$, each of which containing multiple experiments with different focuses.

- Suite A tests learning-parameters;
- Suite B analyzes the planning-parameters;
- Suite C focuses on investigating norms-sets and CTDs;
- Suite D experiments with deontic conflicts, alternative formulations of the norm-reasoning component, and also with simulations of some paradoxes in ASP;
- Lastly, Suite E analyzes the enforcing aspects.

Both Suite A and Suite B have additional results from Bayesian optimizations. Each has their own hypotheses and builds on previous suites.

4.2 Chart-Types

Each experiment produced plots based on data collected during training across the episodes (labeled 'training-results') and afterwards during evaluation (called 'final-results'). The plotted data is averaged over the repetitions. There are twelve chart types, namely:

1. $returns\_training$
2. $returns\_final$

[1] `https://github.com/Gnosling/FrozenLake_DeonticASP`; accessed on May 2, 2025

3. *runtimes_training*
4. *runtimes_final*
5. *steps_slips_explorations_plannings_training*
6. *steps_slips_final*
7. *violations_training*
8. *violations_final*
9. *states_training*
10. *states_final*
11. *states_enforced*
12. *overview_tables*

The plotted data is collected from the target policy, with the exception of explorations and plannings; these values track how often the exploration or the planner was triggered. Because the target does neither use a planner nor behaves ϵ-greedy, this data origins from the behavior policy. Depending on the chart type, the plots are line-plots, box-plots, heat-maps or violin-charts. For example, Figure 4.1 shows the evolution of the returns of the target policy across the training episodes.

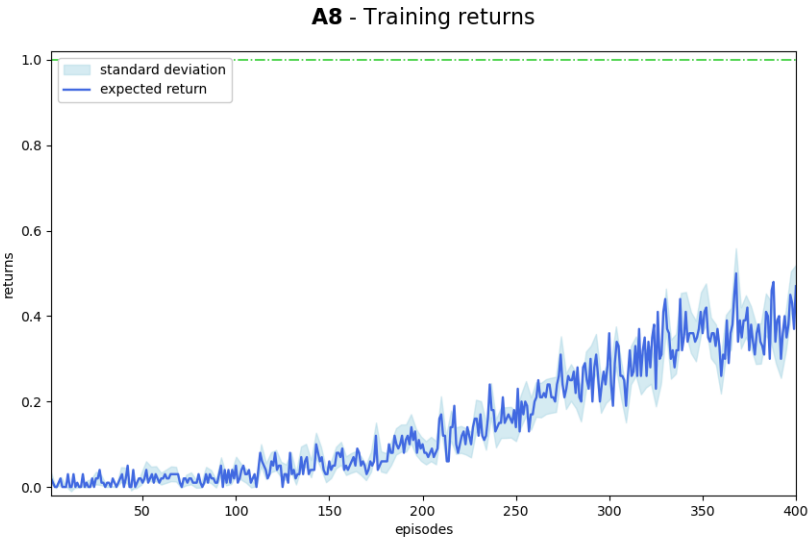

Figure 4.1: Returns of A8 (training)

Moreover, the paths explored by the target policy are collected and displayed in heat-maps (see Figure 4.2). Here each cell corresponds to a respective tile of the FrozenLake level, and the action chosen most often is written down. The action labeled *None* indicates that no actions was performed in this state. All holes and the goal have *None*, because they terminate the episode. The number in each cell shows the average number of visits of that tile; the number on the goal tile equals the expected return. The map highlights the areas that were explored based on the number of visits, and the learned path is displayed through the action sequence starting from the initial tile. For reference Figure

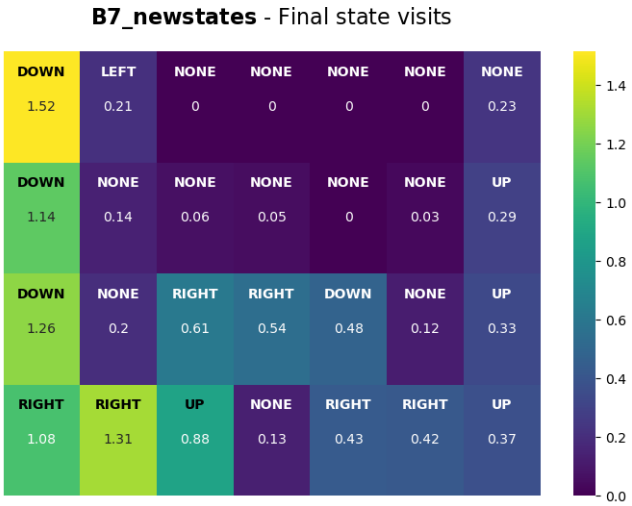

Figure 4.2: Explored states of B7_newstates_states_final

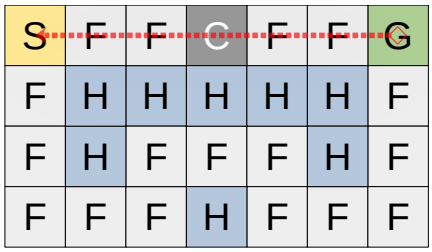

Figure 4.3: FrozenLake-Level: 7x4C (used in B7)

4.3 displays the structure of this level.

The final violations are plotted as a violin charts (see Figure 4.4). These violations are collected from 100 target policies that are evaluated 100-times each, resulting in 10000 entries for every norm. This data is assumed to be normally distributed, except for the norm **notReachedGoal** because it contains only binary values. The final violations can be mirrored by the violations experienced after the post-training enforcement. The chart uses both distributions as one half of the 'violin' for each norm. Although the values are discrete, the area is smoothed into curves for better representation.

Furthermore, the distributions are tested for significant differences. Because the population variance is unknown and only sample variances can be computed, *t-tests* were applied. A *t-test* is used to determine whether there is a significant difference between the means of two groups, assuming that the data is approximately normally distributed.

In particular, *Levene's test* with median-center is performed to check the homogeneity of variances between the two groups [BF74]. Levene's test evaluates the equality of variances

making it suitable for testing whether the variability in each group is homogeneous or consistent. If the variances are not homogeneous, a *two-sampled t-test* is applied, which compares the means of the two independent groups assuming unequal variances. On the other hand, if the variances are homogeneous, *Welch's t-test* is used [Bl47].

The norm **notReachedGoal** is tested by the *chi-squared test* due to its binary structure [Yat34]. The chi-squared test compares the observed frequencies in each category with the expected frequencies under the null hypothesis of no association between the variables. Given that the norm **notReachedGoal** represents a binary outcome (either the goal was reached or not), this test is well-suited for categorical data. If the respective test indicates significant differences between the groups, the label of the respective norm is highlighted in red. An *alpha*-value of 0.001 is used for all tests. Since the data size is large, a small alpha value helps to highlight only the most interesting differences and reduces the chance of false positives.

For example, in Figure 4.4, the distributions for norm **occupiedTraverserTile** and others are significantly different according to the t-test, whereas **notReachedGoal**, **leftSafeArea**, and **didNotReturnToSafeArea** are not.

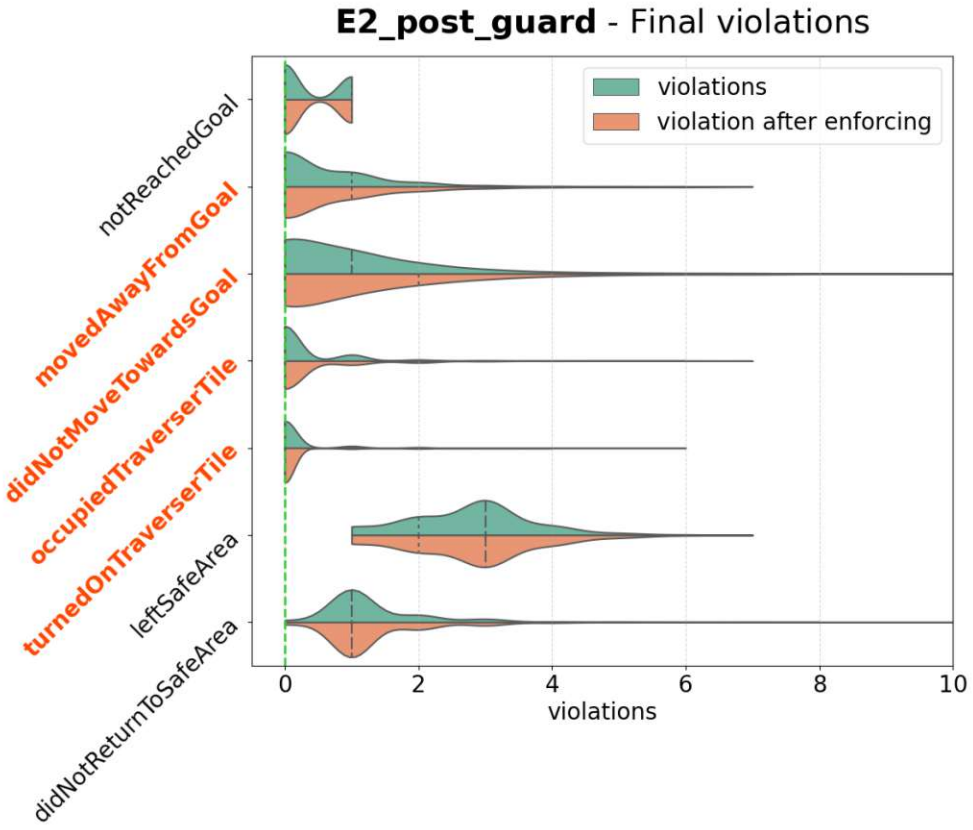

Figure 4.4: Violations of E2_traverser_guard_violations_final (final+enforced)

4.3 A*

These experiments tested the RL-parameters and yielded an initial baseline. Before defining specific configurations, the parameter values were improved using *Bayesian optimization*, with the goal of maximizing the expected return after evaluating the final target policies. As previously mentioned, all results, including detailed data and plots, can be found within the GitHub repository.

4.3.1 Hypotheses

By testing RL-parameters on the FrozenLake, we can make some predictions about their expected values. For instance, when the number of episodes and steps is fixed, the results on larger levels tend to be worse than those on smaller ones. Additionally, the reverse Q-learning method is better suited for environments where rewards are only given at the goal. For A^*, there are five hypotheses:

$H1$ *discount* and *learning-rate* have little impact due to the reward structure
$H2$ reverse q-learning dominates normal q-learning
$H3$ the learned paths prioritize safer routes
$H4$ target policies do not yield solutions for larger levels
$H5$ suitable initialization strategies speed up learning

4.3.2 Results

The Bayesian optimizations were performed on three different levels, namely $4x4\_A$, $6x4\_A$ and $8x8\_A$. Initially, the experiments focused on episodes, the use of reverse Q-learning, discounts, and a constant learning rate for the first two levels. Afterward, dynamic learning rate techniques were tested, followed by the evaluation of different initialization strategies. For each experiment, 20 repetitions and 50 evaluations per target policy were conducted. For instance, Listing 4.2 shows the expected returns of testing some parameters on $4x4\_A$. The best 20 trials all used reversed q-learning, while the discount and learning-rate varies around plausible means. The same can be observed from the other levels. Additionally, a constant learning-rate dominated both the linear_decay and exponential_decay for level $4x4\_A$. Lastly, the initialization strategy *distance* outperformed the others and yielded returns up to 0.7.

```
1 Top trials and their parameters:
2 Trial 25; Value: 0.496; Parameters: {'episodes': 252, 'discount':
    0.8395475115779703, 'reversed_q_learning': True, 'learning_rate':
    0.18815556046386858};
3 Trial 132; Value: 0.475; Parameters: {'episodes': 279, 'discount':
    0.8466259397519311, 'reversed_q_learning': True, 'learning_rate':
    0.20397812380507369};
4 Trial 162; Value: 0.473; Parameters: {'episodes': 249, 'discount':
    0.8422325222133493, 'reversed_q_learning': True, 'learning_rate':
    0.43197792671208635};
```

```
Trial 86; Value: 0.468; Parameters: {'episodes': 251, 'discount':
    0.9189617029401612, 'reversed_q_learning': True, 'learning_rate':
    0.13530255045951342};
Trial 902; Value: 0.465; Parameters: {'episodes': 246, 'discount':
    0.9357989284024836, 'reversed_q_learning': True, 'learning_rate':
    0.11970478397839024};
```

Listing 4.2: Excerpt from
`bayesian_result_of_objective_for_RL_params_L4_1_···`

For level $6x4\_A$, the results generally yielded low returns of less than 0.1. However, initialization strategy $safe$ boosted even the returns noticeable from the other strategies. The $safe$-strategy also had a positive effect on the values of the level $8x8\_A$.

Considering the results of the Bayesian optimizations, default configurations were designed on levels without norm-specifics. All experiments used a constant *learning-rate* of 0.3, a *discount* of 0.99, reversed q-learning, no panning, 100 repetitions and 100 evaluations per final target policy. An example is given in Listing 4.3.

```
"A3": {"repetitions": 100, "episodes": episodes_4x4, "max_steps":
    max_steps_4x4, "evaluation_repetitions": 100,
    "frozenlake": {"name": "FrozenLake4x4_A", "traverser_path": "4
        x4_A", "slippery": True},
    "learning": {"norm_set": None, "epsilon": epsilon, "
        initialization": "distance", "reversed_q_learning": True, "
        discount": 0.99, "learning_rate": 0.3, "learning_rate_strategy
        ": "constant", "learning_decay_rate": None},
    "planning": {"norm_set": 1, "delta": None, "strategy": "
        no_planning", "planning_horizon": None, "reward_set": 2},
    "deontic": {"norm_set": 0, "evaluation_function": None},
    "enforcing": None,
}
```

Listing 4.3: Configuration: A3

Config	Level	Return	Episodes	Episode of Satisfaction	Initialization
A1	$3x3\_A$	0.75	30	7	distance
A2	$4x4\_A$	0.17	50	-	zero
A3	$4x4\_A$	0.59	50	35	distance
A4	$6x4\_A$	0.01	150	-	safe
A5	$6x4\_B$	0.01	150	-	distance
A6	$7x4\_A$	0.2	175	-	safe
A7	$7x4\_A$	0.18	175	-	zero
A8	$7x4\_C$	0.25	175	-	zero
A9	$8x8\_A$	0.42	400	350	safe
A10	$8x8\_A$	0.2	400	-	zero

Table 4.1: Overview of results from A*

The most relevant results of suite A are displayed in Table 4.1. Only configurations $A1$, $A3$ and $A9$ reached a satisfaction point and yielded average returns of more than 0.4, whereas all others failed to learn their respective level. By comparing $A2$ with $A3$ and $A9$ with $A10$ the usage of good initialization strategies significantly benefits the results. However, if the strategy does not completely define a successful path of the level, it yields no significant improvement or worse even a decrease in the returns. For example, configs $A6$ and $A7$ have nearly the same results and only differ by expected randomness.

In more detail, the configurations of level $4x4\_A$ (see A3) did produce different results. The received returns of $A2$ (see Figure 4.5) show a slow but steady increase over the episodes, whereas the ones from $A3$ (see Figure 4.6) have a higher starting point and a strong increase in the early episodes, which stagnates around episode 20. Moreover, the state-visits of both are shown in Figure 4.7 and Figure 4.8. Based on the visits, no clear path was learned for $A2$. Although on average many target policies tend towards the solution, they failed to stabilize individually. As a result, the path constructed by most preferred actions beginning from the start tile, does reach the goal. However, the number of visits reveals that single policies did not follow this path. In contrast to that, the heat-map of $A3$ is consistent with preferred actions and state-visits, showing that the policies did indeed learn the optimal path individually.

Some levels failed to learn a successful policy. $A4$ on level $6x4\_A$ (see A6) yielded final returns of almost 0. The heatmap of both training and final policies (Figure 4.9 and Figure 4.10) failed to reach the goal tile. Although exploration during the training phase discovered the beginnings of the right path, the final target policies did not learn these. Moreover, the initialization strategies influenced the learned path in a few configurations, such as $A9$ and $A10$ on level $8x8\_A$ (see A12). $A9$ used the $safe$-strategy and $A10$ used $zero$-initialization. The first one yielded a path across the diagonal that completely

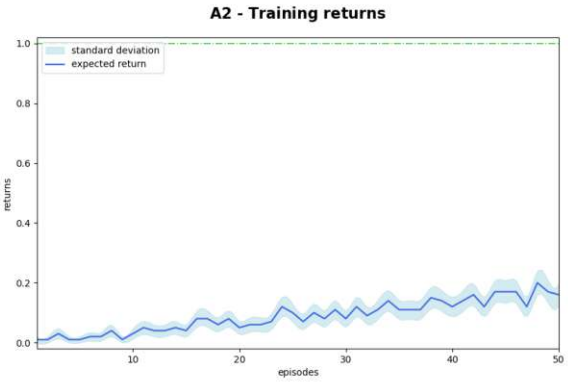

Figure 4.5: Training returns of $A2$

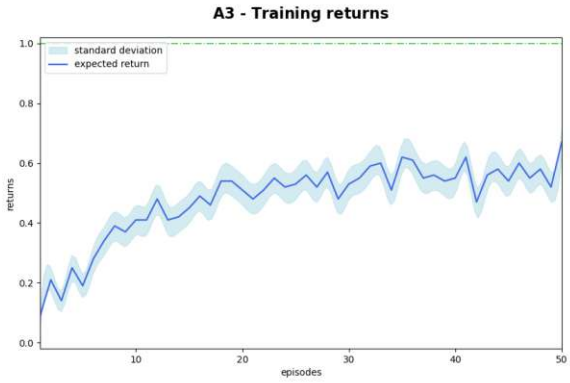

Figure 4.6: Training returns of $A3$

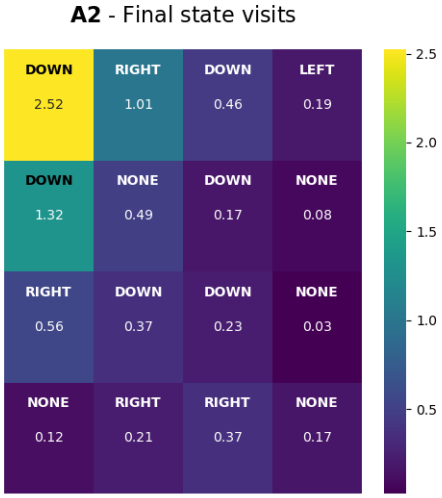

Figure 4.7: Final state visits of $A2$

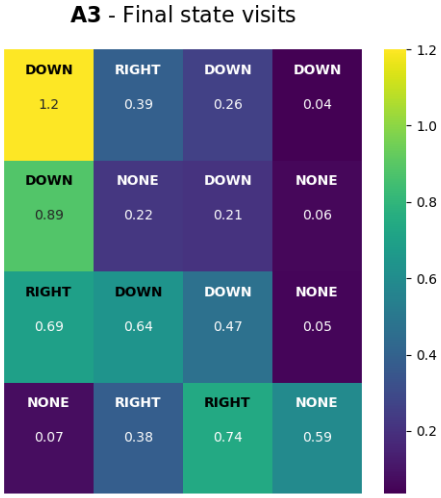

Figure 4.8: Final state visits of $A3$

avoids the cracked tiles in the traverser-area, whereas the latter one learned a page along the edges through the area of the traverser, leading to lower returns.

Figure 4.9: Training state visits of $A4$

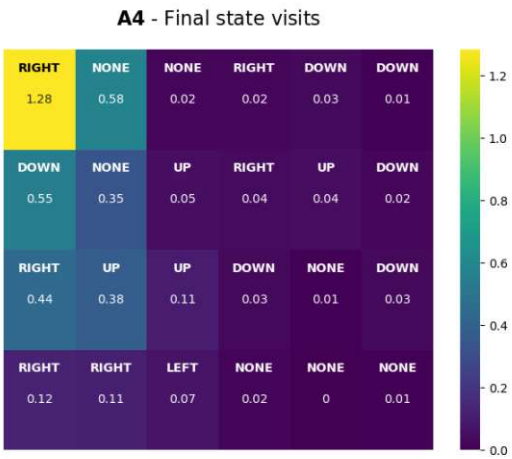

Figure 4.10: Final state visits of $A4$

Figure 4.11: Final state visits of $A9$

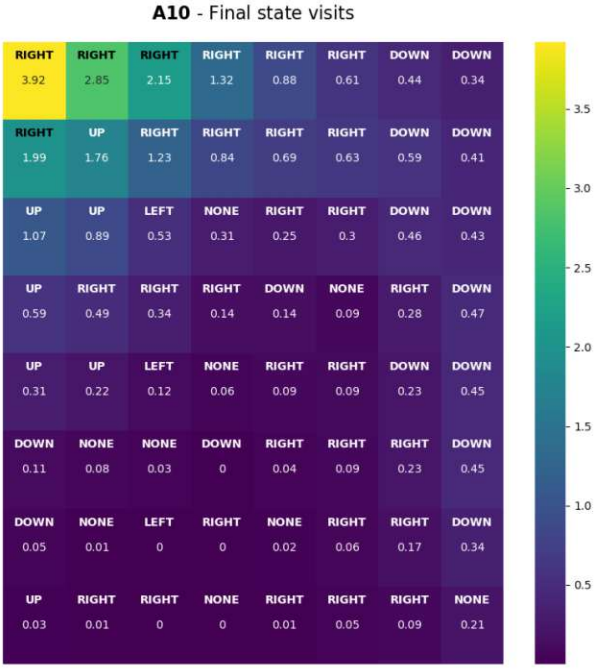

Figure 4.12: Final state visits of $A10$

4.3.3 Summary

The A^* experiments showed the impact of the RL-parameters. Some had negligible impact and others were significant for the learning process. Additionally, increasing complexity in the levels yielded lower returns, and for many configurations pure RL was not sufficient to solve the level. These experiments offered suitable parameter values for the subsequent experiment suites—for instance, reverse Q-learning is now fixed as part of the baseline configuration.

Based on the produced data, the hypotheses can be assessed:

$H1$ *discount* and *learning-rate* have little impact due to the reward structure
→ **Confirmed**, the experiments showed no significant difference.

$H2$ reverse q-learning dominates normal q-learning
→ **Confirmed**, learning speed was tremendously increased in all experiments.

$H3$ the learned paths prioritize safer routes
→ **Rejected**, many experiments did not yield any feasible solution, while others learned shortest paths that might be less safe. Only a minority learned safer routes.

$H4$ target policies do not yield solutions for larger levels
→ **Partially confirmed**, complexity is a larger obstacle than level-size, since $8x8\_A$ was still learned.

$H5$ suitable initialization strategies speed up learning
→ **Confirmed**, learning speed was significantly increased if the strategy is suited for the respective level.

4.4 B*

These experiments tested the ASP-parameters and refined the previous baseline. Similar to A^*, the parameters values were improved using *Bayesian optimization*. Moreover, all planning processes were designed to avoid violating *notReachedGoal*, ensuring a focus on successfully solving the environment.

4.4.1 Hypotheses

The hypotheses for B^* reflect expectations about the effectiveness of planning in RL. $H1$ assumes that incorporating planning will outperform purely model-free approaches, as planning can anticipate future states more effectively. $H2$ builds on this, expecting that increased planning effort leads to better outcomes by reducing uncertainty. $H3$ states that the main benefits during learning will happen right at the beginning and later planning will be ineffective, as the state-action values would be already sufficiently learned. Lastly for $H5$, we expect that planning with good initializations will yield the best outcome. For B^*, there are five hypothesis:

- $H1$ planning dominates model-free RL
- $H2$ the more planning is done, the better the outcome will be
- $H3$ early planning will suffice to improve learning
- $H4$ *full_planning* and *plan_for_new_states* will not yield significantly different results
- $H5$ planning with suitable initialization will yield the best results

4.4.2 Results

Like in A^*, the Bayesian optimizations were performed on levels $4x4\_A$, $6x4\_A$ and $8x8\_A$. Initially, learning rates were tested without planning but with ϵ-greedy exploration. Next, different planning strategies with varying deltas were evaluated. For example, *delta_greedy_planning* has a chance of *delta* to trigger. Lastly, combinations of initialization and planning strategies were examined. All tests used at least 5 repetitions and 50 evaluations per target policy, with the planning horizon fixed for each level to be at least the size of the minimum path. For example, Listing 4.4 shows the expected returns for different parameter settings on $6x4\_A$. Out of the best 20 trials, 8 used full planning, 7 planning for new states, 4 *delta_decaying_planning*, and 1 *delta_greedy_planning*.

Similar to the results of the Bayesian optimization from A^*, different learning rate strategies showed no significant impact on the outcomes. Additionally, the value of ϵ was not particularly relevant within a plausible range. The values of *delta* were optimized towards more planning, i.e. for greedy planning high *delta*'s were preferred and lower values for decay-planning. Initialization strategies had an underwhelming influence when combined with planning. The best results often contained a mix of both effective and

ineffective initialization strategies for the given level. High planning generally overruled initializations during training, thus the strategy became less relevant for optimization.

Based on these results, the values of ϵ, discount and learning rate remained unchanged, while all strategies were tested on the configurations from A^*.

```
1 Top trials and their parameters:
2 Trial 6; Value: 0.572; Parameters: {'planning_strategy': '
     full_planning'};
3 Trial 47; Value: 0.556; Parameters: {'planning_strategy': '
     full_planning'};
4 Trial 64; Value: 0.552; Parameters: {'planning_strategy': '
     delta_decaying_planning', 'delta_decaying': 2.6413502688454443e
     -05};
5 Trial 12; Value: 0.536; Parameters: {'planning_strategy': '
     plan_for_new_states'};
6 Trial 36; Value: 0.536; Parameters: {'planning_strategy': '
     full_planning'};
```

Listing 4.4: Excerpt from `bayesian_result_of_objective_for_ASP_params_L6_2_...`

Config	Level	Return	Episodes	Episode of Satisfaction	Diff-Return to no planning
B1_newstates	$3x3\_A$	0.76	30	7	+ 0.01
B2_greedy	$4x4\_A$	0.62	50	25	+ 0.45
B2_decay	$4x4\_A$	0.62	50	15	+ 0.45
B2_newstates	$4x4\_A$	0.63	50	15	+ 0.46
B2_full	$4x4\_A$	0.63	50	15	+ 0.46
B3_greedy	$4x4\_A$	0.63	50	20	+ 0.04
B3_decay	$4x4\_A$	0.62	50	15	+ 0.03
B3_newstates	$4x4\_A$	0.62	50	15	+ 0.03
B3_full	$4x4\_A$	0.62	50	15	+ 0.03
B4_greedy	$6x4\_A$	0.45	150	45	+ 0.44
B4_decay	$6x4\_A$	0.37	150	20	+ 0.36
B4_newstates	$6x4\_A$	0.47	150	30	+ 0.46
B4_full	$6x4\_A$	0.48	150	25	+ 0.47

Continued on next page

Config	Level	Return	Episodes	Episode of Satisfaction	Diff-Return to no planning
B5_greedy	$6x4\_B$	0.09	150	-	+ 0.08
B5_decay	$6x4\_B$	0.27	150	60	+ 0.26
B5_newstates	$6x4\_B$	0.32	150	100	+ 0.31
B5_full	$6x4\_B$	0.32	150	60	+ 0.31
B6_greedy	$7x4\_A$	0.56	175	30	+ 0.36
B6_decay	$7x4\_A$	0.55	175	15	+ 0.35
B6_newstates	$7x4\_A$	0.61	175	20	+ 0.41
B6_full	$7x4\_A$	0.61	175	15	+ 0.41
B7_greedy	$7x4\_C$	0.41	175	60	+ 0.16
B7_decay	$7x4\_C$	0.43	175	120	+ 0.18
B7_newstates	$7x4\_C$	0.15	175	20	- 0.1
B7_full	$7x4\_C$	0.15	175	10	- 0.1
B8_greedy	$8x8\_A$	0.58	400	250	+ 0.16
B8_decay	$8x8\_A$	0.48	400	200	+ 0.06
B8_newstates	$8x8\_A$	0.57	400	100	+ 0.15
B8_full	$8x8\_A$	0.58	400	100	+ 0.16
B9_greedy	$8x8\_A$	0.58	400	150	+ 0.38
B9_decay	$8x8\_A$	0.54	400	50	+ 0.34
B9_newstates	$8x8\_A$	0.54	400	60	+ 0.34
B9_full	$8x8\_A$	0.54	400	50	+ 0.34

Table 4.2: Overview of results from B*

The most relevant results are shown in Table 4.2. The last column compares the return to the respective value from A^*. The configuration $B1$ has similar yields as $A1$. $B2 - B6$ outperformed $A2 - A7$ significantly and (with exception of $B5\_greedy$) managed to learn the shortest paths on all levels. On the one hand *greedy* and *decay* improved the results of $B7$, on the other hand full planning and planning for new states did not increase the success. Lastly $B8 - B9$ has increased returns compared to $A9 - A10$ but the learned paths are equal.

Configuration $B2$ used no initialization and $B3$ used *distance*. However, the results are

indifferent. For instance, Figures 4.13 and 4.14 have similar learning curves. The initial values had no noticeable impact for any planning strategy. Furthermore, the differences in returns of $B1$ with $A1$ and $B3$ with $A3$ are very little, indicating that good initializations can produce optimal results.

Moving on, exponential decay approximates no triggering of the planning component in later episodes. Figure 4.15 shows how often plannings, explorations and slips were done. The decay increases and planning decreases as the episodes go on, while exploration increases as a result of more Q-table lookups. Furthermore, the learned path of the target policy initially becomes shorter, stagnates, but eventually increases again. This indicates that the learning process diverges, leading the agent to learn a less optimal path. Figure 4.16 reveals a consistent decrease of returns on all repetitions for later episodes. There are also other configurations (like $B6$), where decaying planning produced comparable results.

Figure 4.13: Training returns of $B2$ (greedy)

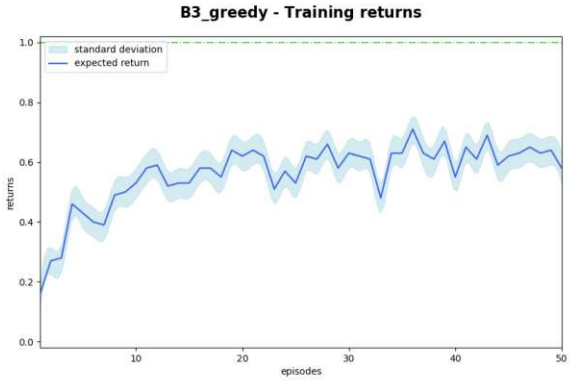

Figure 4.14: Training returns of $B3$ (greedy)

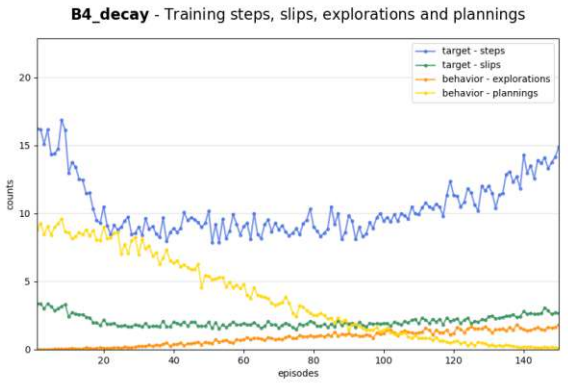

Figure 4.15: Steps $B4$ (decay)

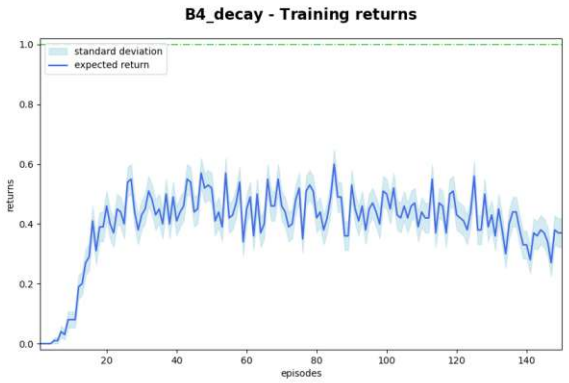

Figure 4.16: Training returns of $B4$ (decay)

On level $6x4\_B$ (see Figure 4.19), the planning component learned a suboptimal path with an extra step. In Figure 4.17, the first action is moving right, but the next one is

moving left and then down. Because the traverser starts on a frozen tile, the planner initially computes the shortest path across the cracked tiles. However, in the subsequent step, the traverser moves up onto the cracked tile, prompting the planner to revise the path and detour onto a longer route. This behavior is learned by all final target policies of $B5$.

Furthermore, the planning component failed to avoid cracked tiles on level $7x4\_C$ (see Figure 4.20). The traverser moves along the first row from right to left, while the agent moves from left to right. However, both players would meet in the middle on a cracked tile, thus failing the level. An optimal policy would involve spending one extra step before reaching the crack to avoid the collision. The planner, however, prioritizes shortest paths and thereby falls into this trap. Moreover, since greedy and decaying planning are triggered less often, they managed to solve the level and yielded higher returns than no planning.

Figure 4.17: Final state visits of $B5$ (newstates)

Figure 4.18: Final state visits of $B7$ (full)

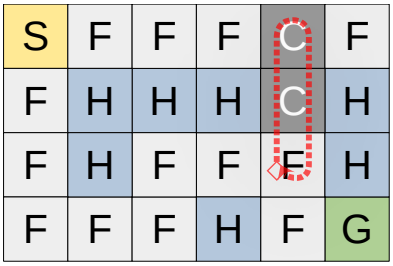

Figure 4.19: FrozenLake level $6x4\_B$

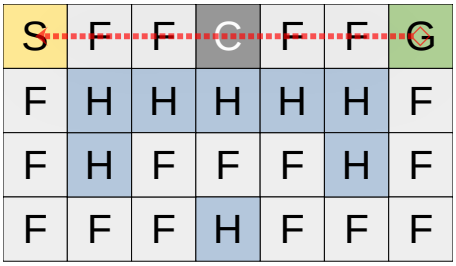

Figure 4.20: FrozenLake level $7x4\_C$

4.4.3 Summary

The B^* experiments demonstrated the impact of ASP-planning on learning the policies. In general, the planning significantly boosted the learning and prioritizes shortest paths.

High usage of plannings overruled any exploration, whereas too little planning resulted in suboptimal policies. Even worse, decaying planning lost its benefits once the chance to trigger became too low, leading to a decrease in the quality of target policies during learning. Additionally, the planner faced difficulties in handling dynamic objects (i.e. the traverser). In each step the traverser's position is assumed to be stationary, causing the planning to overlook potential risks in approaching it. This suggests that the used model may be too simplistic to effectively solve the levels.

Based on the produced data, the hypotheses can be assessed:

$H1$ planning dominates model-free RL
→ **Confirmed**, in general the planning boosted both the speed and quality of learning to reach optimal polices much faster. It can be expected that a more sophisticated model will yield even better results.

$H2$ the more planning is done, the better the outcome will be
→ **Partially confirmed**, increasing planning does increase results, but only up to a certain threshold after which more planning becomes less effective.

$H3$ early planning will suffice to improve learning
→ **Rejected**, the decaying planning has frequent activation in the beginning, yet it underperformed compared to other strategies on most levels.

$H4$ $full\_planning$ and $plan\_for\_new\_states$ will not yield significantly different results
→ **Confirmed**, although $full\_planning$ tends to converge slightly faster, there were no significant differences.

$H5$ planning with suitable initialization will yield the best results
→ **Confirmed**, these combinations had the best policies, but the impact of planning on good initialization was relatively low.

4.5 C*

This suite focuses on testing different norm-sets and evaluations (see 3.1.5). The configurations of B^* are used with *plan_for_new_states* as a baseline. Unlike before, Bayesian optimization is not applied here, as these experiments optimize violations at different levels alongside the return.

4.5.1 Hypotheses

By incorporating deontic reasoning into the framework, we expect the agent to successfully learn and apply CTDs ($H1$), resulting in a reduced number of norm violations compared to agents without norm-awareness ($H2$). When the shortest path conflicts with relevant norms, we assume the agent will instead learn to follow a longer but norm-compliant route ($H3$). Moreover, we expect the agent to learn to avoid the traverser when appropriate norms are used ($H4$). Finally, the framework is expected to prioritize the most critical norms, keeping their violations to a minimum ($H5$).

For C^*, there are five hypothesis:

$H1$ the agent learns CTDs successfully
$H2$ the violations of trained norms are lower than the violations without norm-reasoning
$H3$ if the shortest path is violating relevant norms, then a more lawful and longer path is learned
$H4$ avoidance of traverser is learned if the respective norms are used
$H5$ the most important norms have minimum violations

4.5.2 Results

Config	Level	Returns	Not Reached Goal	Occupied Traverser Tile	Turned On Traverser Tile	Missed Presents	Left Safe Area	Did Not Return To Safe Area
B1_newstates	3x3_B	0.76 ± 0.0	0.24 ± 0.0	0 ± 0.0	0 ± 0.0	0 ± 0.0	*n.a.*	*n.a.*
C1_traverser	3x3_B	0.75 ± 0.0	0.25 ± 0.0	0.02 ± 0.13	0.02 ± 0.13	0.98 ± 0.12	*n.a.*	*n.a.*
C1_present	3x3_B	0.75 ± 0.0	0.25 ± 0.0	0.86 ± 0.53	0.86 ± 0.53	0.15 ± 0.35	*n.a.*	*n.a.*
C1_traverser_with_presents	3x3_B	0.75 ± 0.0	0.25 ± 0.0	0.01 ± 0.1	0.01 ± 0.1	0.99 ± 0.1	*n.a.*	*n.a.*
C2_normless	4x4_B	0.61 ± 0.0	0.39 ± 0.0	0.18 ± 0.48	0.1 ± 0.39	*n.a.*	2.76 ± 0.95	1.38 ± 0.8
C2_traverser	4x4_B	0.61 ± 0.0	0.39 ± 0.0	0.22 ± 0.53	0.11 ± 0.41	*n.a.*	2.72 ± 0.97	1.34 ± 0.83
C2_safe	4x4_B	0.85 ± 0.0	0.15 ± 0.0	0.89 ± 0.92	0.34 ± 0.9	*n.a.*	2.29 ± 0.79	0.32 ± 0.59
C2_traverser_with_safe	4x4_B	0.84 ± 0.0	0.16 ± 0.0	0.85 ± 0.82	0.26 ± 0.79	*n.a.*	2.28 ± 0.77	0.37 ± 0.64
C3_rewards	4x4_C	0.58 ± 0.0	0.42 ± 0.0	0.18 ± 0.51	0.11 ± 0.4	0.16 ± 0.37	*n.a.*	*n.a.*
C3_scaling	4x4_C	0.58 ± 0.0	0.42 ± 0.0	0.13 ± 0.57	0.09 ± 0.43	0.15 ± 0.35	*n.a.*	*n.a.*
C3_violations	4x4_C	0.57 ± 0.0	0.43 ± 0.0	0.12 ± 0.55	0.09 ± 0.43	0.15 ± 0.36	*n.a.*	*n.a.*
C3_weakconstrains	4x4_C	0.56 ± 0.0	0.44 ± 0.0	0.14 ± 0.63	0.1 ± 0.48	0.16 ± 0.36	*n.a.*	*n.a.*

Figure 4.21: Overview of $C1, C2, C3$

Config	Level	Returns	Not Reached Goal	Moved Away From Goal	Did Not Move Towards Goal	Occupied Traverser Tile	Turned On Traverser Tile	Left Safe Area	Did Not Return To Safe Area
BS_newsrates	6x4_B	0.32 ± 0.0	0.68 ± 0.0	1.87 ± 1.28	2.57 ± 1.71	0.11 ± 0.32	*0.1 ± 0.31*	*n.a.*	*n.a.*
C4_traverser	6x4_B	0.13 ± 0.0	0.87 ± 0.0	3.37 ± 2.54	4.62 ± 3.45	0.11 ± 0.32	*0.09 ± 0.3*	*n.a.*	*n.a.*
C4_moving	6x4_B	0.31 ± 0.0	0.69 ± 0.0	2.07 ± 1.47	2.9 ± 2.11	0.11 ± 0.32	*0.1 ± 0.31*	*n.a.*	*n.a.*
C4_strictly_moving	6x4_B	0.31 ± 0.0	0.69 ± 0.0	2.05 ± 1.47	2.84 ± 2.05	0.11 ± 0.32	0.1 ± 0.31	*n.a.*	*n.a.*
C4_traverser_with_moving	6x4_B	0.13 ± 0.0	0.87 ± 0.0	3.39 ± 2.56	4.61 ± 3.41	*0.08 ± 0.28*	0.07 ± 0.27	*n.a.*	*n.a.*
B6_newsrates	7x4_A	0.61 ± 0.0	0.39 ± 0.0	0.73 ± 0.92	*n.a.*	*n.a.*	*n.a.*	3.15 ± 1.34	*1.79 ± 0.78*
C5_safe	7x4_A	0.68 ± 0.0	0.32 ± 0.0	1.97 ± 1.31	*n.a.*	*n.a.*	*n.a.*	3.57 ± 1.47	*0.38 ± 0.53*
C5_moving	7x4_A	0.61 ± 0.0	0.39 ± 0.0	0.71 ± 1.05	*n.a.*	*n.a.*	*n.a.*	2.67 ± 1.2	1.78 ± 0.62
C5_safe_first	7x4_A	0.67 ± 0.0	0.33 ± 0.0	2.12 ± 1.34	*n.a.*	*n.a.*	*n.a.*	3.7 ± 1.63	0.45 ± 1.04
C5_moving_first	7x4_A	0.64 ± 0.0	0.36 ± 0.0	0.8 ± 0.97	*n.a.*	*n.a.*	*n.a.*	2.64 ± 1.11	1.56 ± 0.79
C5_equal	7x4_A	0.63 ± 0.0	0.37 ± 0.0	0.82 ± 1.07	*n.a.*	*n.a.*	*n.a.*	2.63 ± 1.12	1.56 ± 0.8
CG_traverser_first	7x4_B	0.4 ± 0.0	0.6 ± 0.0	*n.a.*	*n.a.*	0.53 ± 0.57	0.07 ± 0.26	3.8 ± 2.57	1.32 ± 1.24
CG_safe_first	7x4_B	0.47 ± 0.0	0.53 ± 0.0	*n.a.*	*n.a.*	0.14 ± 0.45	0.04 ± 0.22	4.22 ± 2.39	1.25 ± 0.82
CG_equal	7x4_B	0.4 ± 0.0	0.6 ± 0.0	*n.a.*	*n.a.*	0.52 ± 0.57	0.07 ± 0.26	4.07 ± 2.81	1.3 ± 1.18
B7_greedy	7x4_C	0.41 ± 0.0	0.59 ± 0.0	1.19 ± 1.08	2.97 ± 2.23	0.42 ± 0.77	0.09 ± 0.51	*n.a.*	*n.a.*
C7_normless	7x4_C	0.16 ± 0.0	0.84 ± 0.0	0.41 ± 0.76	1.16 ± 1.54	0.64 ± 0.56	0.02 ± 0.15	*n.a.*	*n.a.*
C7_moving	7x4_C	0.15 ± 0.0	0.85 ± 0.0	0.4 ± 0.76	1.13 ± 1.5	0.64 ± 0.54	0.02 ± 0.15	*n.a.*	*n.a.*
C7_strictly_moving	7x4_C	0.16 ± 0.0	0.84 ± 0.0	0.42 ± 0.76	1.17 ± 1.5	0.63 ± 0.5	0.02 ± 0.17	*n.a.*	*n.a.*
C7_traverser_with_moving	7x4_C	0.23 ± 0.0	0.77 ± 0.0	3.03 ± 1.73	4.24 ± 2.28	0.02 ± 0.24	0.0 ± 0.05	*n.a.*	*n.a.*

Figure 4.22: Overview of $C4, C5, C6, C7$

Config	Level	Returns	Not Reached Goal	Moved Away From Goal	Occupied Traverser Tile	Turned On Traverser Tile	Stole Present	Missed Presents	Left Safe Area	Did Not Return To Safe Area
C8_normless	7x4_D	0.23 ± 0.0	0.77 ± 0.0	2.73 ± 1.73	*n.a.*	*n.a.*	1.49 ± 1.38	1.51 ± 1.38	*n.a.*	*n.a.*
C8_no_presents	7x4_D	0.54 ± 0.0	0.46 ± 0.0	0.32 ± 0.78	*n.a.*	*n.a.*	0 ± 0.0	3 ± 0.0	*n.a.*	*n.a.*
C8_presents	7x4_D	0.25 ± 0.0	0.75 ± 0.0	2.64 ± 1.64	*n.a.*	*n.a.*	1.52 ± 1.38	1.48 ± 1.38	*n.a.*	*n.a.*
C8_moving	7x4_D	0.25 ± 0.0	0.75 ± 0.0	2.59 ± 1.53	*n.a.*	*n.a.*	1.5 ± 1.38	1.5 ± 1.38	*n.a.*	*n.a.*
B8_newstates	8x8_A	0.57 ± 0.0	0.43 ± 0.0	*n.a.*	0.07 ± 0.26	0 ± 0.0	*n.a.*	*n.a.*	2.51 ± 1.05	*2.65 ± 1.7*
C9_traverser	8x8_A	0.5 ± 0.0	0.5 ± 0.0	*n.a.*	0.01 ± 0.08	0 ± 0.0	*n.a.*	*n.a.*	2.81 ± 1.27	*2.78 ± 1.57*
C9_traverser_over_safe	8x8_A	0.43 ± 0.0	0.57 ± 0.0	*n.a.*	0.5 ± 0.5	0 ± 0.0	*n.a.*	*n.a.*	0.98 ± 1.01	0.16 ± 0.93
C10_normless	8x8_B	0.29 ± 0.0	0.71 ± 0.0	*n.a.*	*n.a.*	*n.a.*	*n.a.*	0.64 ± 0.48	4.93 ± 2.59	2.88 ± 2.56
C10_safe	8x8_B	0.8 ± 0.0	0.2 ± 0.0	*n.a.*	*n.a.*	*n.a.*	*n.a.*	2.03 ± 0.18	*1.89 ± 2.18*	0.69 ± 2.33
C10_presents	8x8_B	0.31 ± 0.0	0.69 ± 0.0	*n.a.*	*n.a.*	*n.a.*	*n.a.*	1.32 ± 1.13	*4.83 ± 2.58*	3.07 ± 3.1
C10_safe_with_presents_1	8x8_B	0.83 ± 0.0	0.17 ± 0.0	*n.a.*	*n.a.*	*n.a.*	*n.a.*	2.02 ± 0.14	*1.73 ± 1.4*	0.67 ± 2.39
C10_safe_with_presents_2	8x8_B	0.6 ± 0.0	0.4 ± 0.0	*n.a.*	*n.a.*	*n.a.*	*n.a.*	1.44 ± 0.73	3.51 ± 1.77	0.82 ± 1.45

Figure 4.23: Overview of $C8, C9, C10$

An overview of the results is provided in Figures 4.21, 4.22, and 4.23. Each C-group is compared based on the trained norms and the expected rewards. If the normless variant was already in B^*, it is included in the comparison. Norms that are not included in the configuration are marked as *'not applicable'*. The experiment names include suffixes that indicate the norms used or their assigned priorities. For example, $C5\_safe$ used the $safeArea$-norms, whereas $C5\_moving\_first$ indicates that the $movingTowardsGoal$-norm has the highest evaluation priority. The numbers shown are averages and standard deviations from all evaluations, and the norm-columns list the amount of violations. The returns are contrasted with violations of $notReachedGoal$, and their sum must be close to 1 (small rounding errors might prevent perfect match). Cells marked in yellow represent the lowest value in the column, while those marked in blue represent the highest. In general, if the returns are equal, but the violations differ, then the learned paths were optimized to fulfill more norms, but share similar structures. However, if the returns differ, entirely new paths were likely learned.

On level $4x4\_B$ (see Figure A4), the experiments of $C2$ tested traverser avoidance with preference of safe areas. Configurations $C2\_normless$ and $C2\_traverser$ yielded similar results and also learned the same path, with no significant differences. In contrast, $C2\_safe$ learned a different path along the edges, which also resulted in higher returns. Lastly, $C2\_traverser\_with\_safe$ used all four norms with equal levels, i.e. $level(occupiedTraverserTile, 2)$, $level(turnedOnTraverserTile, 3)$, $level(leftSafeArea, 2)$ and $level(didNotReturnToSafeArea, 3)$. However this configuration produced results similar to those of $C2\_safe$. Figures 4.24, 4.25, 4.26, and 4.27 display the learned path of $C2$ and highlight the differences. Notably, the safer path was the most norm-compliant.

For $C4$, all configurations learned similar policies, except for $C4\_traverser$ and $C5\_traverser\_with\_moving$. These configurations performed worse than the normless variant $B5\_newstates$. When the traverser blocks the path to the goal of level $6x4\_B$ (see 4.19), the planner advises the agent to backtrack every time, whereas the other setups would only go back if there is a cracked tile. However, this overcaution led to worse policies with significant stalling. Although the learned path on average is the same on all norm-sets, the violations in Figure 4.28 and 4.29 show these differences in the moving violations.

The decision between reaching the goal quickly, staying in safe areas, and avoiding the traverser leads to different policies. $C5$ can be grouped into two categories, namely policies that learned the shortest path and those who learned a safer path. Only $C5\_safe$ and $C5\_safe\_first$ used safer paths. Moreover, the top level norm decides the behavior of the agent, for instance $C5\_moving$ and $C5\_moving\_first$ produced nearly identical results. However, the different paths of safer and faster routes are shown in Figures 4.30 and 4.31.

Similar observation regarding safe paths (as seen $C5$) and traverser overcautions (as in $C4$) can be observed in $C6$ on level $7x4\_B$ (see A9) via Figures 4.32 and 4.33. Although the learned path in the latter configuration does not reach the goal, the individual policies may have a solvable path in their behavior, and the averages merge them into the one

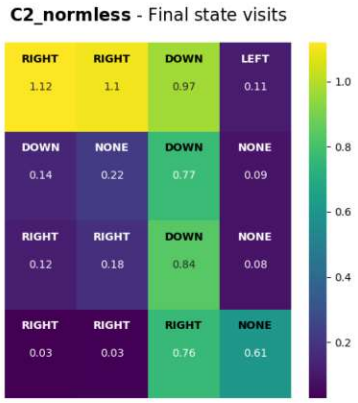

Figure 4.24: Final state visits of $C2$ (normless)

Figure 4.25: Final state visits of $C2$ (traverser)

Figure 4.26: Final state visits of $C2$ (safe)

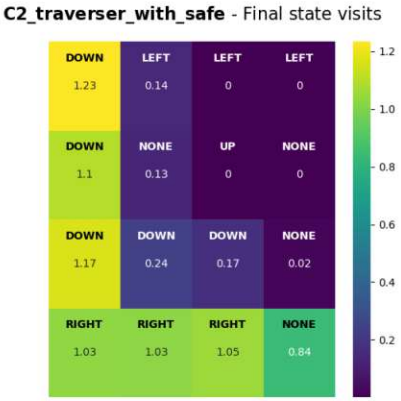

Figure 4.27: Final state visits of $C2$ (traverser with safe)

shown in the figure. Additionally, the slips prevent the agent from getting stuck in a loop (life-locking). Furthermore, configuration $C7\_traverser\_with\_moving$ learned to avoid the traverser, while all other settings failed to learn the detour. Nonetheless, the greedy planner from $B7$ outperforms all configurations in $C7$. Figures 4.34 and 4.35 show the policy behaviors.

Because the presents are handled in the internal rewards, the agent favors to collect them. Level $8x8\_B$ (see A13) contains three presents, whereby the one on the right is in a safe area, the one in the left bottom corner has one hole near it, and the present in the middle is next to three holes. They can be seen as safe, moderate and risky subgoals respectively.

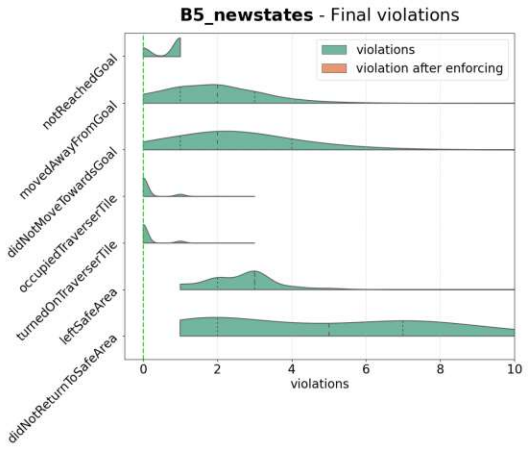

Figure 4.28: Final violations of $B5$ (newstates)

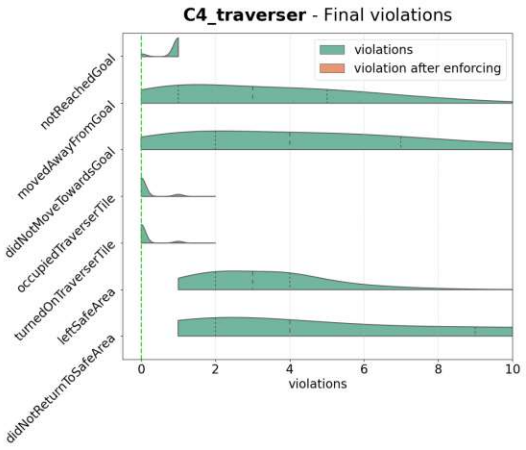

Figure 4.29: Final violations of $C4$ (traverser)

Experiment $C10$ illustrates the different paths the agent learned when using the present and safe-area norms. On the one hand, $C10\_normless$ and $C10\_presents$ learned a path across all presents, although the final returns were lower (see Figure 4.36). On the other hand, $C10\_safe$ and $C10\_safe\_with\_presents\_1$ followed a safer path along the edges, collecting only one present (see Figure 4.37). There are two sets with all three norms (i.e. $C10\_safe\_with\_presents\_1$ and $C10\_safe\_with\_presents\_2$), whereby the first has $level(leftSafeArea, 2)$,$level(didNotReturnToSafeArea, 3)$, $level(missedPresents, 2)$ and the latter $level(leftSafeArea, 2)$, $level(didNotReturnToSafeArea, 3)$, $level(missedPresents, 3)$. By increasing the level of $missedPresents$, the training resulted in a path along three edges that picks up two presents (see Figure 4.38). Thus, three distinct paths were explored with different set of norms, whereby the riskier one pick up more presents and yield less return, while the safer ones pick up less presents and complete the level with a higher chance.

Figure 4.30: Final state visits of $C5$ (moving)

Figure 4.31: Final state visits of $C5$ (safe)

Figure 4.32: Final state visits of $C6$ (safe first)

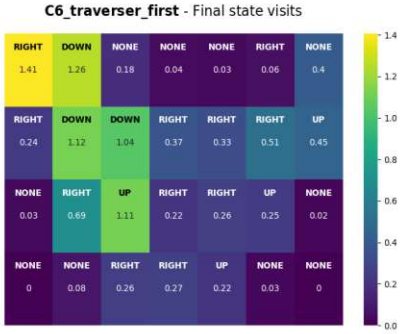

Figure 4.33: Final state visits of $C6$ (traverser first)

Figure 4.34: Final state visits of $B7$ (greedy)

Figure 4.35: Final state visits of $C7$ (traverser with moving)

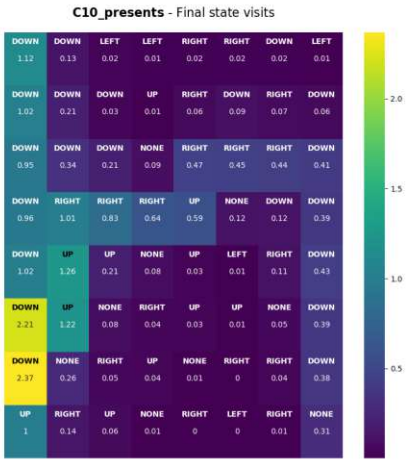

Figure 4.36: Final state visits of $C10$ (presents)

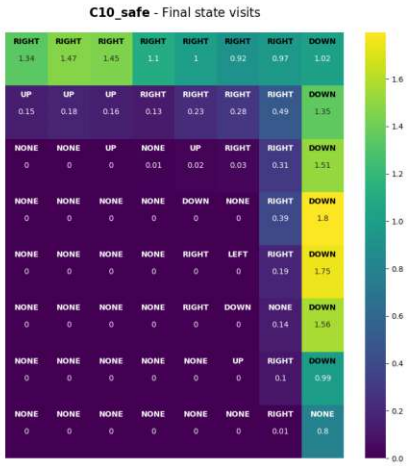

Figure 4.37: Final state visits of $C6$ (safe)

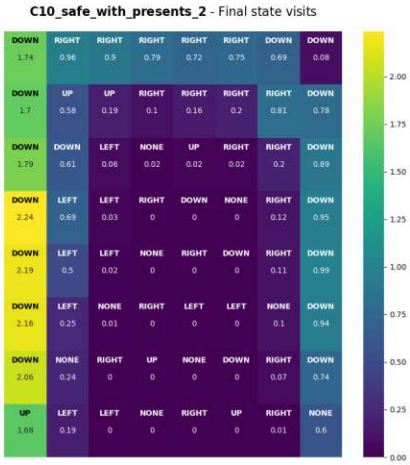

Figure 4.38: Final state visits of $C7$ (safe with presents 2)

4.5.3 Summary

The Suite C^* experiments tested different sets of norms used for planning to alter the agent's behavior. The norms are prioritized by level, and the experiments demonstrated that the most important norm dictates the agents behavior. If multiple norms share the highest priority, the planner optimizes the path by reducing the norm that has the higher chance of being violated. Grouping many norms together without carefully configuring their priorities can lead to unintended side effects, such as completely discarding a high-priority norm to fully reduce the violation of another high-priority norm, rather than balancing both equally. We expand more on this in the Discussion (Chapter 5) and in the reviews of the results for Suite E^*.

Without any norms, the planning component favors the shortest path with most internal subgoal (i.e. the presents). If this is already lawful, then the norm-reasoning did not change the final policy. However, if the generated paths violate norms, the final policies will differ. Depending on the priorities and evaluation strategies, the agent can be pushed to take riskier or more lawful actions. Too lawful behavior can hinder the agent's ability to solve the level and may lead to live-locking due to overcautions. On the other hand, overly risky behavior can distract the agent from his long-term goal and might lead to unnecessary violations. Ultimately, no single strategy is optimal across all level instances.

Based on the produced data, the hypotheses can be assessed:

$H1$ agent learns CTDs successfully
→ **Partially confirmed**, if the levels of CTDs are high enough, then this hold in the experiments.

$H2$ the violations of trained norms are lower than the violations without norm-reasoning
→ **Partially confirmed**, trained norms resulted overall in less violations. Only the *notReachedGoal*-norm could become more violated in some experiments.

$H3$ if the shortest path is violating relevant norms, then a more lawful and longer path is learned
→ **Confirmed**.

$H4$ avoidance of traverser is learned if the respective norms are used.
→ **Partially confirmed**, although the agent learned a stronger avoidance, in some experiments the traverser-tile was still occupied.

$H5$ the most important norms have minimum violations
→ **Partially confirmed**, if many norms have highest priority than only the overall sum is minimized and not individual norms.

4.6 D*

The Suite D^* experiments was concerned with deontic conflicts, alternative formulations of the norm-reasoning component, and also with simulations of some paradoxes in ASP. However, they are not the focus of this work.

The framework is expected to resolve normative conflicts by favoring one interpretation or side ($H1$). It should treat semantically equivalent formulations interchangeable, without altering behavior ($H2$). For example, replacing OA with $F\neg A$ should yied the same result. Moreover, some textbook paradoxes will not cause issues ($H3$), and lastly in the absence of progressing norms, the agent is expected to be disincentivized from reaching the goal ($H4$).

For D^*, there are four hypothesis:

4.6.1 Hypotheses

$H1$ conflicting norms will favor one side and solve the level
$H2$ different formulations of equal normative aspects are not going to have an impact
$H3$ the ASP-planner can resolve Sartre's Dilemma and Ross's Paradox
$H4$ restrictive norms without 'progress'-norms can hinder the agent to learn the level

4.6.2 Results

Config	Level	Returns	Not Reached Goal	Moved Away From Goal	Stole Present	Missed Presents	Left Safe Area	Did Not Return To Safe Area
D1_present	3x3_B	0.74 ± 0.0	0.26 ± 0.0	*n.a.*	0.84 ± 0.37	0.16 ± 0.37	*n.a.*	*n.a.*
D1_no_present	3x3_B	0.75 ± 0.0	0.25 ± 0.0	*n.a.*	0.01 ± 0.08	0.99 ± 0.08	*n.a.*	*n.a.*
D1_both	3x3_B	0.75 ± 0.0	0.25 ± 0.0	*n.a.*	0.85 ± 0.36	0.15 ± 0.36	*n.a.*	*n.a.*
D2_normal	4x4_A	0.63 ± 0.0	0.37 ± 0.0	*n.a.*	*n.a.*	*n.a.*	*n.a.*	*n.a.*
D2_ross	4x4_A	0.62 ± 0.0	0.38 ± 0.0	*n.a.*	*n.a.*	*n.a.*	*n.a.*	*n.a.*
D3_factual	4x4_B	0.6 ± 0.0	0.4 ± 0.0	*n.a.*	*n.a.*	*n.a.*	*n.a.*	*n.a.*
D3_deontic	4x4_B	0.61 ± 0.0	0.39 ± 0.0	*n.a.*	*n.a.*	*n.a.*	*n.a.*	*n.a.*
D4_F(R)_F(M)	7x4_A	0.59 ± 0.0	0.41 ± 0.0	0.85 ± 1.46	*n.a.*	*n.a.*	*n.a.*	*n.a.*
D4_O(R)_F(M)	7x4_A	0.59 ± 0.0	0.41 ± 0.0	0.73 ± 1.12	*n.a.*	*n.a.*	*n.a.*	*n.a.*
D4_F(R)_O(M)	7x4_A	0.6 ± 0.0	0.4 ± 0.0	0.81 ± 0.98	*n.a.*	*n.a.*	*n.a.*	*n.a.*
D4_O(R)_O(M)	7x4_A	0.62 ± 0.0	0.38 ± 0.0	0.79 ± 0.97	*n.a.*	*n.a.*	*n.a.*	*n.a.*
D5_newstates	8x8_B	0.6 ± 0.0	0.4 ± 0.0	*n.a.*	*n.a.*	*n.a.*	1.74 ± 1.84	0.33 ± 0.71
D5_full	8x8_B	0.31 ± 0.0	0.69 ± 0.0	*n.a.*	*n.a.*	*n.a.*	0.75 ± 0.84	0.14 ± 0.44

Figure 4.39: Overview of $D1, D2, D3, D4, D5$

An overview of the results is given in Figure 4.39. In $D1$, there are two possible paths, one with the present and one without. $D1\_no\_present$ learned the path avoiding the present, while $D1\_both$ produced similar results to $D1\_present$, both learned the path with the present (see Figures 4.40 and 4.41). No significant differences were found in experiments $D2$, $D3$ and $D4$. These experiments tested out different formulations in ASP;

e.g. $D2$ tries to simulate Ross's Paradox and $D4$ formulated its norms as prohibitions F and as obligations O in all combinations.

Lastly, $D5$ was trained on level $8x8\_B$ without the *reachingGoal*-norm, causing the agent to prioritize staying in safe areas. However, in order to solve the level it needed to move at least past one hole. The full planner failed to learn an optimal path, whereas planning for new states yielded better results, but still performed worse than $C10\_safe$ (which included avoidance of *notReachedGoal*). The agent in $D5$ ended up cycling within a safe area; the learned paths can be seen in Figures 4.42 and 4.43.

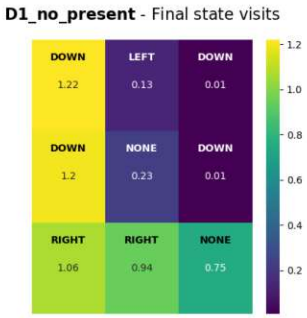

Figure 4.40: Final state visits of $D1$ (no present)

Figure 4.41: Final state visits of $D1$ (both)

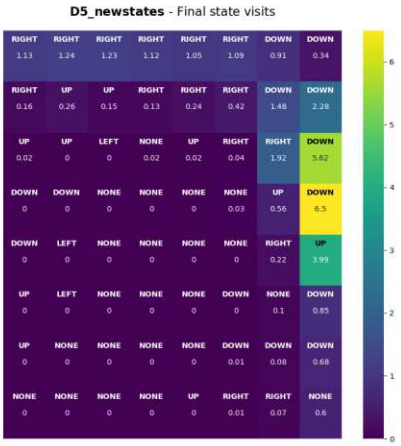

Figure 4.42: Final state visits of $D5$ (new states)

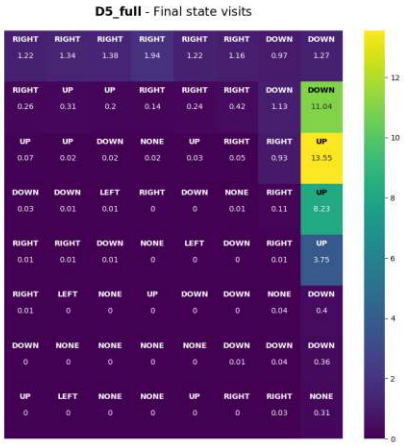

Figure 4.43: Final state visits of $D5$ (full)

4.6.3 Summary

These experiments tested edge cases and the stability of norm-reasoning within the framework. While some paradoxes were simulated, the main CTDs are covered in C^*. The planning component successfully resolved the tested paradoxes and dilemmas, remaining

consistent across different formulations. However, it struggled to learn successful paths when the *reachingGoal*-norm was excluded.

Based on the produced data, the hypotheses can be assessed:

*H*1 conflicting norms will favor one side and solve the level
→ **Confirmed**.

*H*2 different formulations of equal normative aspects are not going to have an impact
→ **Confirmed**, the experiments showed no significant differences.

*H*3 the ASP-planner can resolve Sartre's Dilemma and Ross's Paradox
→ **Confirmed**, the ASP planning managed to resolve conflicts and to avoid deriving unintended norms.

*H*4 restrictive norms without 'progress'-norms can hinder the agent to learn the level
→ **Confirmed**, the learned paths cycle in lawful areas.

4.7 E*

The final experiments of Suite E^* were applying the enforcing strategies (see Subsection 3.1.6). Based on the configuration from C^*, different enforcing methods are applied to generate new policies.

The enforcement strategies are expected to reduce the number of norm violations ($H1$). Applying them to the final target policies may prove more effective than enforcing them only during training ($H2$). From the discussed strategies, guardrailing could restrict the agent from achieving best long-term results ($H3$), while policy fixing might fail to enforce the agent to avoid the traverser entirely ($H4$), and since only full rewards shaping considers all norms, it is expected to result in better norm compliance than the optimal reward shaping ($H5$). Lastly, norm-based initialization strategies are expected to yield the best outcomes overall, as it provides the agent with a form of moral grounding before training begins ($H6$).

4.7.1 Hypotheses

$H1$ Enforcing strategies improve violations, but might harm the returns

$H2$ Applying enforcings after the training phase yields better result than the enforcings during training

$H3$ **Action-Guardrailing** might generate worse returns due to its short term restrictions

$H4$ **Policy-Fixing** fails to avoid traverser completely (similar to the planning component)

$H5$ Full **Reward-Shaping** has better norm-compliance, whereas the optimal one has better returns

$H6$ The norm-based initialization strategies produce the best results with a high returns and low violations

4.7.2 Results

Config	Level	Returns	Returns *Enforced*	Not Reached Goal	Not Reached Goal *Enforced*	Occupied Traverser Tile	Occupied Traverser Tile *Enforced*	Turned On Traverser Tile	Turned On Traverser Tile *Enforced*
E1_training_guard	4x4_B	0.61 ± 0.0	*n.a.*	0.39 ± 0.0	*n.a.*	0.23 ± 0.54	*n.a.*	0.12 ± 0.42	*n.a.*
E1_training_fix	4x4_B	0.67 ± 0.0	*n.a.*	0.33 ± 0.0	*n.a.*	0.81 ± 0.69	*n.a.*	0.26 ± 0.6	*n.a.*
E1_training_hull_shaping	4x4_B	0.6 ± 0.0	*n.a.*	0.4 ± 0.0	*n.a.*	0.21 ± 0.51	*n.a.*	0.11 ± 0.4	*n.a.*
E1_training_opt_shaping	4x4_B	0.6 ± 0.0	*n.a.*	0.4 ± 0.0	*n.a.*	0.22 ± 0.53	*n.a.*	0.12 ± 0.42	*n.a.*
E2_post_guard	4x4_B	0.6 ± 0.0	0.6 ± 0.0	0.4 ± 0.0	0.4 ± 0.0	0.23 ± 0.55	0.28 ± 0.67	0.12 ± 0.43	0.07 ± 0.35
E2_post_fix	4x4_B	0.62 ± 0.0	0.67 ± 0.0	0.38 ± 0.0	0.33 ± 0.0	0.22 ± 0.54	0.87 ± 0.85	0.12 ± 0.42	0.23 ± 0.6
E2_post_hull_shaping	4x4_B	0.6 ± 0.0	0.61 ± 0.0	0.4 ± 0.0	0.39 ± 0.0	0.23 ± 0.57	0.25 ± 0.62	0.12 ± 0.45	0.13 ± 0.5
E2_post_opt_shaping	4x4_B	0.61 ± 0.0	0.62 ± 0.0	0.39 ± 0.0	0.38 ± 0.0	0.22 ± 0.52	0.22 ± 0.51	0.12 ± 0.42	0.11 ± 0.4
E3_guard	6x4_B	0.32 ± 0.0	0.29 ± 0.0	0.68 ± 0.0	0.71 ± 0.0	0.1 ± 0.31	0.12 ± 0.37	0.1 ± 0.3	0.09 ± 0.29
E3_fix	6x4_B	0.32 ± 0.01	0.3 ± 0.01	0.68 ± 0.01	0.7 ± 0.01	0.11 ± 0.33	0.13 ± 0.38	0.1 ± 0.32	0.1 ± 0.3
E3_opt_shaping	6x4_B	0.32 ± 0.0	0.33 ± 0.0	0.68 ± 0.0	0.67 ± 0.0	0.11 ± 0.32	0.11 ± 0.32	0.1 ± 0.31	0.1 ± 0.31
E3_hull_shaping	6x4_B	0.33 ± 0.0	0.32 ± 0.0	0.67 ± 0.0	0.68 ± 0.0	0.11 ± 0.32	0.11 ± 0.32	0.1 ± 0.31	0.1 ± 0.31
E3_hull_state_function	6x4_B	0.32 ± 0.0	*n.a.*	0.68 ± 0.0	*n.a.*	0.11 ± 0.32	*n.a.*	0.1 ± 0.3	*n.a.*
E3_hull_action_penalty	6x4_B	0.33 ± 0.0	*n.a.*	0.67 ± 0.0	*n.a.*	0.11 ± 0.32	*n.a.*	0.1 ± 0.3	*n.a.*

Figure 4.44: Overview of $E1, E2, E3$

Config	Level	Returns	Returns *Enforced*	Occupied Traverser Tile	Occupied Traverser Tile *Enforced*	Turned On Traverser Tile	Turned On Traverser Tile *Enforced*	Left Safe Area	Left Safe Area *Enforced*	Did Not Return To Safe Area	Did Not Return To Safe Area Enforced
E4_guard	7x4_B	0.18 ± 0.0	0.07 ± 0.0	0.19 ± 0.47	0.21 ± 0.44	0.04 ± 0.26	0.02 ± 0.14	2.56 ± 2.01	7.69 ± 4.92	1.52 ± 1.0	0.87 ± 0.71
E4_fix	7x4_B	0.17 ± 0.0	0.09 ± 0.0	0.21 ± 0.61	0.66 ± 0.51	0.07 ± 0.46	0.02 ± 0.13	2.59 ± 2.22	2.91 ± 2.0	1.54 ± 1.28	0.31 ± 0.48
E4_opt_shaping	7x4_B	0.17 ± 0.0	0.19 ± 0.0	0.23 ± 0.49	0.23 ± 0.48	0.06 ± 0.27	0.05 ± 0.27	2.51 ± 2.03	2.57 ± 2.17	1.54 ± 0.94	1.5 ± 0.93
E4_full_shaping	7x4_B	0.09 ± 0.0	0.06 ± 0.0	0.1 ± 0.36	0.04 ± 0.32	0.03 ± 0.17	0.01 ± 0.1	3.17 ± 2.6	3.94 ± 3.02	1.8 ± 1.2	0.64 ± 1.05
E4_init_state_function	7x4_B	0.05 ± 0.0	*n.a.*	0.05 ± 0.26	*n.a.*	0.01 ± 0.12	*n.a.*	1.46 ± 1.36	*n.a.*	1.24 ± 0.63	*n.a.*
E4_init_action_penalty	7x4_B	0.25 ± 0.0	*n.a.*	0.58 ± 0.63	*n.a.*	0.08 ± 0.28	*n.a.*	3.58 ± 2.5	*n.a.*	0.77 ± 0.93	*n.a.*
E5_guard	7x4_B	0.51 ± 0.0	0.11 ± 0.0	0.78 ± 0.56	0.5 ± 0.53	0.14 ± 0.34	0.03 ± 0.16	2.52 ± 1.11	5.51 ± 4.29	1.7 ± 0.72	0.7 ± 0.74
E5_fix	7x4_B	0.52 ± 0.0	0.09 ± 0.0	0.77 ± 0.56	0.67 ± 0.5	0.13 ± 0.34	0.02 ± 0.13	2.57 ± 1.23	2.88 ± 1.9	1.73 ± 0.71	0.3 ± 0.48
E5_opt_shaping	7x4_B	0.52 ± 0.0	0.53 ± 0.0	0.77 ± 0.55	0.75 ± 0.57	0.13 ± 0.31	0.13 ± 0.34	2.52 ± 1.08	2.76 ± 1.6	1.72 ± 0.72	1.79 ± 0.72
E5_full_shaping	7x4_B	0.5 ± 0.0	0.55 ± 0.0	0.76 ± 0.56	0.64 ± 0.58	0.13 ± 0.32	0.11 ± 0.31	2.56 ± 1.26	3.36 ± 2.2	1.67 ± 0.71	1.87 ± 0.7
E5_init_state_function	7x4_B	0.52 ± 0.0	*n.a.*	0.76 ± 0.56	*n.a.*	0.13 ± 0.33	*n.a.*	2.53 ± 1.12	*n.a.*	1.69 ± 0.7	*n.a.*
E5_init_action_penalty	7x4_B	0.52 ± 0.0	*n.a.*	0.77 ± 0.55	*n.a.*	0.13 ± 0.34	*n.a.*	2.54 ± 1.16	*n.a.*	1.71 ± 0.68	*n.a.*
E6_guard	7x4_B	0.39 ± 0.0	0.27 ± 0.0	0.56 ± 0.57	0.56 ± 0.55	0.08 ± 0.26	0.05 ± 0.21	3.92 ± 2.87	5.64 ± 3.94	1.38 ± 1.45	0.81 ± 0.85
E6_fix	7x4_B	0.43 ± 0.0	0.09 ± 0.0	0.56 ± 0.58	0.66 ± 0.51	0.08 ± 0.28	0.02 ± 0.13	3.57 ± 2.36	2.86 ± 1.93	1.31 ± 0.9	0.3 ± 0.48
E6_opt_shaping	7x4_B	0.39 ± 0.0	0.48 ± 0.0	0.5 ± 0.57	0.53 ± 0.58	0.07 ± 0.25	0.09 ± 0.28	3.96 ± 2.87	3.65 ± 2.34	1.26 ± 1.26	1.38 ± 0.88
E6_full_shaping	7x4_B	0.3 ± 0.0	0.4 ± 0.0	0.46 ± 0.53	0.14 ± 0.38	0.03 ± 0.17	0.02 ± 0.14	4.14 ± 2.81	6.22 ± 3.9	0.82 ± 0.83	1.0 ± 0.97
E6_init_state_function	7x4_B	0.42 ± 0.0	*n.a.*	0.4 ± 0.54	*n.a.*	0.05 ± 0.22	*n.a.*	4.18 ± 2.63	*n.a.*	1.15 ± 0.85	*n.a.*
E6_init_action_penalty	7x4_B	0.51 ± 0.0	*n.a.*	0.13 ± 0.34	*n.a.*	0.01 ± 0.08	*n.a.*	5.27 ± 2.87	*n.a.*	0.61 ± 0.7	*n.a.*

Figure 4.45: Overview of $E4, E5, E6$

Config	Level	Returns	Returns *Enforced*	Occupied Traverser Tile	Occupied Traverser Tile *Enforced*	Turned On Traverser Tile	Turned On Traverser Tile *Enforced*	Left Safe Area	Left Safe Area *Enforced*	Did Not Return To Safe Area	Did Not Return To Safe Area Enforced
E7_guard	8x8_A	0.46 ± 0.0	0.46 ± 0.0	0.46 ± 0.5	0.47 ± 0.5	0 ± 0.0	0 ± 0.0	0.88 ± 0.9	0.97 ± 1.14	0.44 ± 1.37	0.09 ± 0.31
E7_fix	8x8_A	0.46 ± 0.0	0.25 ± 0.0	0.46 ± 0.5	0.53 ± 0.5	0 ± 0.0	0 ± 0.0	0.86 ± 0.88	2.33 ± 1.01	0.44 ± 1.4	0.24 ± 0.44
E7_full_shaping	8x8_A	0.46 ± 0.0	0.72 ± 0.0	0.46 ± 0.5	0.22 ± 0.41	0 ± 0.0	0 ± 0.0	0.87 ± 0.9	0.92 ± 0.98	0.42 ± 1.27	0.31 ± 1.16
E7_full_action_penalty	8x8_A	0.52 ± 0.0	*n.a.*	0.41 ± 0.49	*n.a.*	0 ± 0.0	*n.a.*	0.91 ± 0.95	*n.a.*	0.17 ± 0.61	*n.a.*
E8_guard	8x8_A	0.55 ± 0.0	0.54 ± 0.0	0.42 ± 0.49	0.42 ± 0.49	0 ± 0.0	0 ± 0.0	0.67 ± 0.87	0.69 ± 0.89	0.11 ± 0.73	0.05 ± 0.26
E8_fix	8x8_A	0.54 ± 0.0	0.24 ± 0.0	0.41 ± 0.49	0.54 ± 0.5	0 ± 0.0	0 ± 0.0	0.68 ± 0.85	2.33 ± 1.03	0.11 ± 0.72	0.24 ± 0.43
E8_full_shaping	8x8_A	0.54 ± 0.0	0.74 ± 0.0	0.42 ± 0.49	0.22 ± 0.42	0 ± 0.0	0 ± 0.0	0.67 ± 0.83	0.75 ± 0.86	0.12 ± 0.86	0.09 ± 0.58
E8_full_action_penalty	8x8_A	*0.58 ± 0.0*	*n.a.*	*0.38 ± 0.48*	*n.a.*	0 ± 0.0	*n.a.*	*0.72 ± 0.97*	*n.a.*	*0.05 ± 0.24*	*n.a.*
E9_guard	8x8_A	0.54 ± 0.0	0.53 ± 0.0	0.41 ± 0.49	0.44 ± 0.5	0 ± 0.0	0 ± 0.0	0.68 ± 0.87	0.36 ± 0.63	0.12 ± 0.82	0.03 ± 0.2
E9_fix	8x8_A	0.54 ± 0.0	0.47 ± 0.0	0.41 ± 0.49	0.49 ± 0.5	0 ± 0.0	0 ± 0.0	0.68 ± 0.85	0.71 ± 0.75	0.11 ± 0.7	0.05 ± 0.24
E9_full_shaping	8x8_A	0.56 ± 0.0	0.74 ± 0.0	0.4 ± 0.49	0.22 ± 0.41	0 ± 0.0	0 ± 0.0	0.7 ± 0.88	0.74 ± 0.85	0.12 ± 0.85	0.1 ± 0.64
E9_full_action_penalty	8x8_A	0.59 ± 0.0	*n.a.*	0.37 ± 0.48	*n.a.*	0 ± 0.0	*n.a.*	0.7 ± 0.81	*n.a.*	0.04 ± 0.21	*n.a.*
E10_guard	8x8_A	0.53 ± 0.0	0.47 ± 0.0	0.42 ± 0.49	0.37 ± 0.48	0 ± 0.0	0 ± 0.0	0.67 ± 0.84	1.19 ± 1.99	0.1 ± 0.77	0.63 ± 1.89
E10_fix	8x8_A	0.54 ± 0.0	0.22 ± 0.0	0.41 ± 0.49	0.4 ± 0.49	0 ± 0.0	0 ± 0.0	0.68 ± 0.86	2.16 ± 1.17	0.12 ± 0.79	0.2 ± 0.41
E10_full_shaping	8x8_A	0.55 ± 0.0	0.74 ± 0.0	0.41 ± 0.49	0.22 ± 0.42	0 ± 0.0	0 ± 0.0	0.69 ± 0.88	0.78 ± 0.89	0.13 ± 0.88	0.11 ± 0.75
E10_full_action_penalty	8x8_A	0.56 ± 0.0	*n.a.*	0.39 ± 0.49	*n.a.*	0 ± 0.0	*n.a.*	0.83 ± 1.61	*n.a.*	0.07 ± 0.29	*n.a.*

Figure 4.46: Overview of $E7, E8, E9, E10$

The results are shown in Figures 4.44, 4.45, and 4.46. Note that the initialization strategies do not require post-training enforcement; therefore, the corresponding columns are marked as *'not applicable'*. $E1$ and $E2$ used the same sets of norms for planning and enforcing, but tested different enforcement strategies. $E1$ applied enforcing during the training, i.e. directly in the behavior policy, whereas in $E2$. enforcing was applied to the target policies after the training was completed. In both groups, only the 'policy-fixing' strategy produced different results and all other enforcings yielded similar outputs. There was no significant difference between training-enforcement and post-enforcement. The fixing strategy changed the agent's path and increased the expected return, but it did not improve the encountered violations (see Figures 4.47, 4.48, and 4.49).

All configurations of $E3$ produced nearly identical results. The planning component already determines the optimal path for level $6x4\_B$ and the enforcings did not change the policy in any significant way. The generated data is almost identical to that of $B5\_newstates$.

$E4 - E6$ tested different norms during planning while enforcing only the CTDs together with reaching the goal. More concretely, $E4$ had no planning at all, $E5$ incorporated the simple norms $occupiedTraverserTile$ and $leftSafeArea$, and $E6$ employed all norms related to the traverser and safe areas in the planner. $E4$ generally resulted in low returns and failed to learn the level. Although $E4\_init\_action\_penalty$ and $E4\_fix$ learned feasible paths, the agent failed to avoid the traverser and ultimately crashed on the cracked tile (see Figures 4.50 and 4.51). Furthermore, $E4\_guard$ generated a policy in which the agent stalls before crossing the section with the traverser, preventing it from reaching the goal (see Figure 4.52). Although $E5$ and $E6$ used planning, the guardrailing and fixing strategies led to similar issues, disrupting the already learned paths. Additionally, compared to $E5$, $E6$ achieved generally lower returns but less violations, with $E6\_init\_action\_penalty$ having the best tradeoff with high returns and low violations.

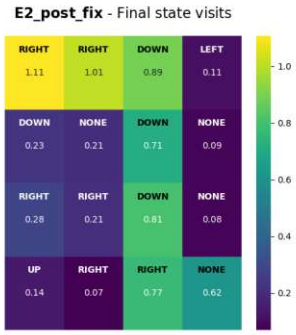

Figure 4.47: Final state visits of $E2\_post\_fix$

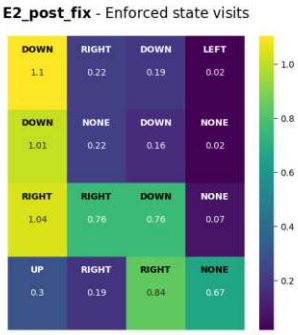

Figure 4.48: Enforced state visits of $E2\_post\_fix$

Next, groups $E7 - E10$ were all tested on level $8x8\_A$ (see Appendix A12) with the traverser and safe area norms. Specifically, $E7$ used simple norms for planning and

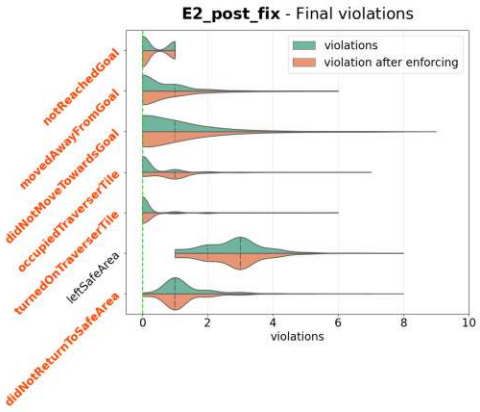

Figure 4.49: Violations of $E2\_post\_fix$

Figure 4.50: Enforced state visits of $E4$ (action-penalty)

Figure 4.51: Enforced state visits of $E4$ (fixing)

Figure 4.52: Enforced state visits of $E4$ (guard)

enforced the CTDs, $E8$ planned for all and enforced CTDs, $E9$ used all norms for both planning and enforcing, and lastly $E10$ is the same as $E9$ with adding the moving-towards-goal norm in the enforcing. In all setups, there were no turnings on a traverser-tile and the fixing disrupted the already learned paths, leading to overall worse policies, similar to the previous experiments (see Figure 4.55). Guardrailing decreased violations of CTDs, but not significantly for simple norms, and the learned paths remained similar to those from the planning. For example, Figures 4.53 and 4.54 show the differences in paths, where the guarding has lower visits in not relevant areas. Moving on, full reward shaping yielded the highest returns and the lowest occupation of traverser-tiles, but did not produce a new path structure. Because the level contains cracked tiles, policies that avoid the traverser are more successful in reaching the goal. The violations are shown in Figure 4.56. Furthermore, the shaping approach overall outperforms the initialization with action-penalty in both rewards and violations. Although it slightly decreases the violations of meeting the traverser, the initialization is insufficient to learn an avoidance. Additionally, this setup misled the agent during trainings into local optimas in early episodes, for example Figure 4.57 highlights a wrongly explored path on the left middle

part of the level which was abandoned by the final policies (Figure 4.58).

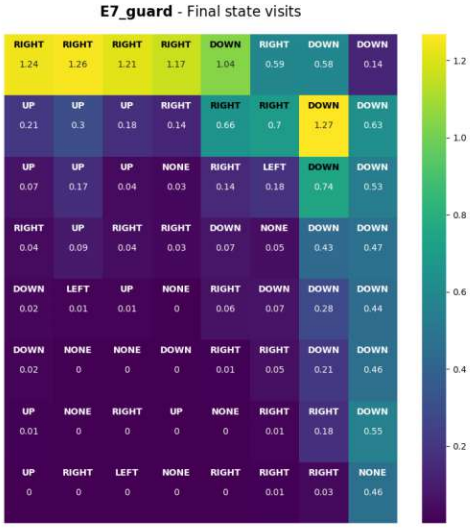

Figure 4.53: Final state visits of $E7$ (guard)

Figure 4.54: Enforced state visits of $E7$ (guard)

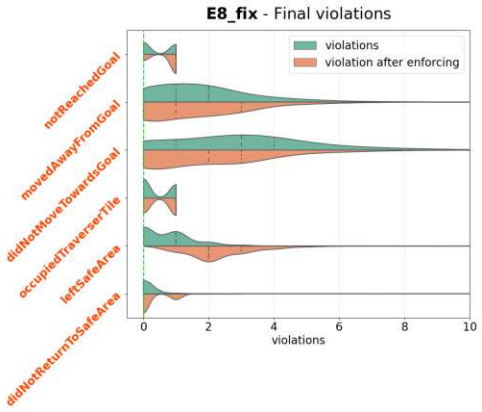

Figure 4.55: Violations of $E8$ (fix)

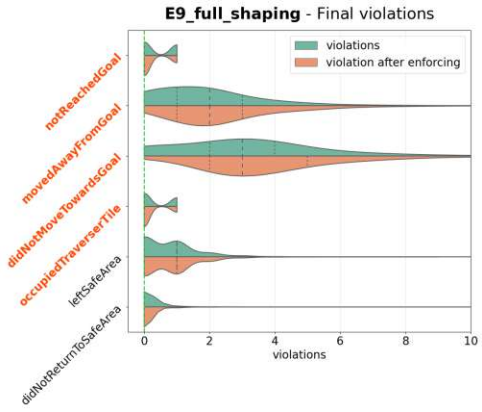

Figure 4.56: Violations of $E9$ (full-shaping)

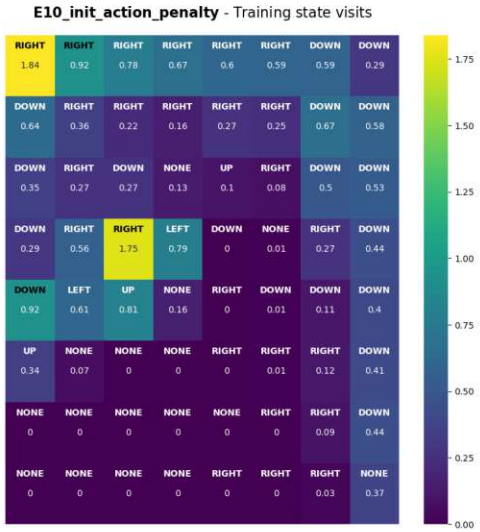

Figure 4.57: Training state visits of $E10$ (action-penalty)

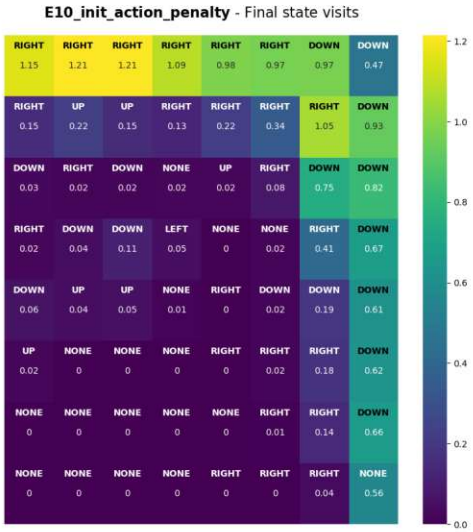

Figure 4.58: Final state visits of $E10$ (action-penalty)

4.7.3 Summary

The last group of experiments demonstrated the impact of different enforcement strategies on the policy. The enforcement stages had no noticeable distinctions between the enforcing during the training phase and afterwards. Enforcing an already optimal policy neither improved nor worsened the behavior. When the policy is not optimal, there is an interplay between planning and enforcement. Generally, using more norms in both planning and enforcement results in the agent acquiring more lawful behaviors, though certain setups can lead to worse outcomes.

The guardrailing of actions produced mixed results. On some levels, it had no significant impact, on others it led to overcautious behavior causing the agent to fail the level, and in certain environments the guard resulted in a noticeable improvement in violations. This enforcement type appears to be highly dependent on the applied norms and the structure of the level.

Furthermore, policy fixing overall significantly worsened the policies. Certain issues mentioned in B^* regarding full planning were repeated when using the fixing strategy. Because this enforcement still relies on the same planner and is activated every time a violation is detected, it may completely overrule the RL-component.

Next, shaping of rewards never decreased the returns, in fact a slight increase was consistently generated in the experiments. However, the results for violations were mixed. The optimal shaping ignored the CTDs (as explained in Section 3.1.6) but still enforced the 'reachingGoal'-norm, but the difference between the two versions seems to be insignificant though. Full shaping managed to learn and avoid the traverser path and yielded the highest return on level $8x8\_A$ compared to all other experiments on that level.

Lastly, the norm-based initialization strategies generally increased returns and mostly decreased violations, with the action-penalty dominating the state-values. However, they might misled the agent during training into local maxima. As the training continued, the agent used planning and exploration to correct these insufficient initializations. Nonetheless if the strategy is suitable, it can speed up learning and potentially yield better results.

Based on the produced data, the hypotheses can be assessed:

*H*1 Enforcing strategies improve violations, but might harm the returns
→ **Partially confirmed**, this generally holds for the majority of strategies, but it does not hold for policy-fixing at all and a few other setups also worsened violations.

*H*2 Applying enforcings after the training phase yields better result than the enforcings during training
→ **Inconclusive**, these experiments showed no significant differences, but more data is needed to assess this thesis.

H3 **Action-Guardrailing** might generate worse returns due to its short term restrictions
→ **Confirmed**, overcautious behaviors were generated by the guard.

H4 **Policy-Fixing** fails to avoid traverser completely (similar to the planning component)
→ **Confirmed**.

H5 Full **Reward-Shaping** has better norm-compliance, whereas the optimal one has better returns
→ **Inconclusive**, these experiments had mixed results of the shaping variants, but more data is needed to assess this thesis.

H6 The norm-based initialization strategies produce the best results with high returns and low violations
→ **Partially confirmed**, in some experiments the action-penalty initialization dominates, while in others the full-shaping of rewards is superior. However both generally yield good results across the experiments.

CHAPTER 5

Discussion

The experiments utilized different components and strategies to learn several levels of the extended FrozenLake. To recap, A^* is pure RL, B^* combined RL and ASP, C^* added norm-reasoning, D^* tested stability of the framework, and lastly E^* explored enforcement strategies for norms.

The pure RL-component was unable to learn larger FrozenLake levels. Because the FrozenLake only provides a reward of +1 on the goal tile, there is effectively no feedback for any other action, causing that the agent is exploring aimlessly until it by coincidence reaches the goal tile. Additionally, RL-parameters such as learning rate and discount factor had no significant impact either. The usage of reverse Q-learning significantly boosted learning. It updates all state-action pairs along any path reaching the goal within a single episode, immediately propagating the solving reward to all relevant actions. Furthermore, proper initialization strategies greatly accelerated learning. However, these strategies generated the best results if they already solved the environment and created an optimal policy without any further training. Misleading strategies, on the other hand, might hinder or delude the agent.

After adding planning, the results were overwhelmingly improved, but the internal model of FrozenLake lacks knowledge of the traverser's movement. Because no prediction rules are defined, the model assumes the traverser remains fixed. This was intentionally designed with the hope that explorations and learning will mitigate this limitation. However, B^* revealed that the policies failed to avoid the traverser. Thus the model is insufficient for efficiently solving some levels, and a more sophisticated model would likely produce better results for these configurations. Different prediction strategies could improve the situation, such as a memory to store previous visited tiles by the traverser to which it could cycle back, or assuming the traverser repeats its last movement. Regarding non-determinism, the model did not face major issues, only favoring shortest paths distracts the planner from safer routes. Nonetheless, as long as the expected outcome has the highest probability, the model can be used to plan a suitable path.

Both the ASP-planning and the RL-components suggest actions in current states, with the planning strategies defining which suggestion should be used. Full planning could immediately define and stabilize the optimal policy, but it might fall into the pitfalls of the insufficient modeling and favors full exploitation over exploration. On the other hand, no planning increases the training time required for RL to stabilize the policy and aims for full exploration in the FrozenLake, although dynamic aspects can be learned.

The mixed approaches of δ-greedy, δ-decaying and *new_states*-planning balanced both extremes differently. The first and last methods seemed to be the most consistent. Although, more planning led to better results, there is a threshold beyond which it becomes less effective or even negligible. In our experiments, planning for *new_states* generally yielded the best results. Moreover, δ-decaying planning failed to balance ASP with RL. During training, the policies improved initially but eventually worsened as the chance of using planning approached zero. This indicates that planning dominates in the beginning , while learning takes precedence at the end of the training However, due to the increasing exploration and the lack of rewards in FrozenLake, this results in losing the benefits of early planning. Using more strategies with different values of parameter δ could also generate additional insight into the interaction of ASP-planning and RL-Learning.

Norm reasoning was added in the planning component by using weak constraints of varying priorities, with the relevant norms and their levels injected into the model according to the specific setup. The planner optimizes the highest priority first, and the most important norm dictates the agents behavior. If multiple norms share the highest priority, the planner optimizes the path by reducing the norm that is more likely to be violated. However, all of these priorities must be manually configured. Using all norms with equal weights would not yield usable results, as the semantic meaning would be lost, and the agent would simply minimize the overall sum. Even with different priorities, instead of reducing individual violations of norms with the same relevance, the agent focuses only on reducing the sum of violations for each group, without balancing individual norms. Furthermore, using all norms together could misrepresent some intensions. Even if the priorities and evaluations are properly configured, there is no guarantee that the agent will consider all intended moral aspects during optimization.

The pure planning component favors the shortest paths with most internal subgoals, without considering safer alternatives. The norm reasoning impacts these paths if violations occur. The used norms and their levels influence the agent to behave more risky or safer. Excessively lawful behavior could hinder the agent in solving the level and might lead to live-locking due too overcautions. Conversely, overly risky behavior could distract the agent from his long-term goals and may result in unnecessary violations of norms. This trade-off cannot be universally solved across all levels, and typically, a balanced approach yields the best results in terms of both returns and violations.

Moreover, adding norms to avoid the traverser had only an underwhelming impact. The agent learned to not approach the traverser when not necessary, but the movement of the traverser was still not learned due to the already discussed limitations of the planning

model. Norms regarding the safe areas were generally learned but the agent might become overcautious if no completed safe path to the goal exists. Progress-related violations (i.e. $didNotMoveTowardsGoal$, $movedAwayFromGoal$, $didNotReachGoal$) are necessary to prevent the agent from life-locking in lawful areas that do not lead to the goal. These norms have different semantics from simply not moving back, focusing on not moving closer, or not reaching the goal by the end of the episode. The moving norms did not impact planning, but they did yield different results. Due to the slipping mechanic, the agent might slide into the current tile. For instance, moving up on the upper edge of the level would not change the agent's position. The agent's model, however, is unaware of this movement and cannot plan for not moving at all, and thus both norms generated the same plans. The reaching-goal norm might mislead the agent and fail to resolve stalling situations consistently. For example, consider tow paths, A and B, both are leading to the goal. If the agent is on path A and the dynamic obstacle is blocking path A but B is feasible, the agent plans to go to B. In the next step, if the traverser moves also to B as well, the agent plans to return to A. Although the $didNotReachGoal$-violation has higher priority, it fails to resolve this situation, because it is only violated at the last step of the planning. The norms related to the subgoals (i.e. the presents) functioned as expected. The planning model already has internal rewards for collecting presents, so the law of taking all presents did not influence the agent's default behavior. However forbidding the agent from taking or missing some presents allowed these subgoals to be compared with other norms. Depending the chosen evaluation and priorities, the agent's desire for presents can be adjusted in any direction.

Although complex norms or morals cannot be fully quantified by values or categorized into priority groups, simple rules and CTDs can be expressed using these approaches in a planning model. However, hard-coding these rules is not feasible for abstract morals due to numerous potential triggers. For example, a norm like 'Be nice to other players' would need to account for for all possible cases in any model, specifying exactly what actions are considered unfriendly. The norms used in the FrozenLake experiments are straightforward, but the framework limits their complexity. More abstract norms were not tested in our experiments due to this restriction. Furthermore, the first experimental groups did not explore different treatment of certain norms or using RL to learn the norms directly. Thus additional norm-reasoning strategies, i.e. the enforcements, were developed at a later stage and tested in E^*.

The D^* experiments are not the primary focus of this work, and their main purpose was to test stabilization of the framework, to briefly explore simulations of alternative formulation of deontic operators and some minor paradoxes. However, more detailed studies on these aspects can be found in other papers. In this work, these experiments revealed that Telingo, within our framework, indeed behaves as expected in integrating deontic reasoning into ASP-solvers as described in Section 2.3.4.

The idea behind the enforcing strategies was to improve policies concerning critical norms. The different enforcements can be applied on behavior and target polices independently (see Section 3.1.6). In the experiments this is referred to as enforcing during training

or post training. However, on the tested levels, no significant differences were observed between the two approaches. Since the enforcements were originally intended to alter already trained policies, there are only few experiments directly comparing their active phases. There would be potential for more configuration to analyze differences though. Moreover the smaller levels are too simple to efficiently test the enforcements. Larger levels provided more interesting data to analyze, whereas enforcing did not alter already optimal policies on simpler levels.

The guardrailing of actions had mixed results, with varying impacts on returns and violations. The aim of this strategy is to restrict actions in critical states, while still allowing the agent to choose from the remaining options. Depending on the configuration, the guard generated outcomes ranging from excessive caution to minor reductions in violations. Furthermore, it lacks long-term checks and sophisticated evaluations of norms (it simply allows actions with the fewest violations). Thus optimal paths might become restricted, especially if many norms are enforced. The guard produced better result in experiments, where fewer norms were considered and where the norms did not lead to overly cautious behavior. Although the setup determines its quality and positive or negative impacts, guardrailing is a straightforward norm-enforcing component that can be integrated into any interaction cycle of agent and environment.

The next enforcing type, fixing of policy, did not work as intended. Nearly all policies were downgraded by this strategy. It essentially acts as a planer that checks for any violation and, upon detection, triggers the planning component to output a corrected action. However, when using many norms, there is a high chance that some norm will always be violated, triggering the planning process every time and completely overriding the learned policies. Additionally, the fixing strategy suffers from the same issues as full planning due to the insufficient model (e.g. overcaution, failure to avoid traverser). Using distinct sets of norms for checking and for fixing could improve this approach, for instance, by checking for violations of critical norms only, while planning the correction with consideration of all relevant norms. Furthermore, this approach significantly increased the inference time of the polices. While other setups perform look-ups in the Q-table, this one triggers the ASP-solver after training is complete. Although this did not create major issues in our experiments, due to the implemented caches and simplicity of the FrozenLake, policy fixing becomes increasingly inefficient as the environment scale increases.

Reward shaping uses violations to either define the lawfulness of a position by the optimal shaping or penalizes state-action-pairs via full shaping. In our experiments, these were employed to modify the feedback from the environment and as initialization of the Q-Table. Optimal shaping cannot express CTDs, so the experiments used it either for a collection of norms or only to enforce the goal-reaching norm. Both variants slightly increased returns, but the experiments on smaller levels did not reveal major differences between them. Initially, it was expected that the optimal shaping would yields better returns, while the full shaping would result in fewer violations. Although CTDs cannot be expressed with the state structure used in our experiments, and the optimal shaping only considered simple norms, the full shaping did not result in significantly lower returns.

However, further configurations are needed to investigate this in more detail. Moreover, unlike all other tested norm-reasoning approaches, shaping does not dictate which norms the agent is supposed to learn. Instead, the feedback is an evaluated sums of adhered-to norm or their violations. The evaluation is manually configured based on the level of norms, but the agent does not posses any inherent norm-reasoning itself. Since the feedback comes directly from the environment, there are no internal model errors, and the previously described issues do not arise. In fact, full shaping succeeded in learning dynamic environments, yielding the best traverser-avoidance among all other experiments.

By using the rewards functions as initialization-strategies, the agent was expected to behave more lawfully from the beginning. However, by blindly following this setup, the agent might be mislead into local maxima. Eventually, the agent will break free due to planning exploitation, learning exploration, and non-determinism. In general the initializations produced the best policies only if their computed values resulted in an optimal path. Similar to the non-norm-based initializations, no strategy solves all levels optimally. The agent can be sped up or slowed down during learning, and its resulting behavior may comply with different norms than those used to preset the Q-table. Furthermore, pre-computing state-actions pairs is only feasible in discrete environment with limited states and actions. Without sophisticated optimizations, this approach becomes increasingly inefficient with higher scalability. In continuous environments with comprehensive actions (such as robots with full joint movements), norm-based initializations could be unmanageable.

CHAPTER 6

Related Work

This chapter presents related experiments, frameworks, and key insights into the integration of RL, ASP, and deontic reasoning. Several of the reviewed works directly inspired aspects of our methodology, while others highlight possible extensions and enhancements to our framework.

6.1 Norm Compliance in RL

Schiehl [Sch23] introduced a framework that integrates normative compliance within Lexicographic Multi-Objective Reinforcement Learning (*LMORL*) agents. In their approach, the objectives are prioritized lexicographically, with the most critical goal, such as normative compliance, optimized first, followed by secondary objectives like task completion or efficiency. The framework incorporates a norm satisfaction score to evaluate how well the agent's actions align with established norms. This score is then included as one of the key objectives in the lexicographic hierarchy. Additionally, reward shaping is applied through penalties for norm violations. While the technical implementation differs, both this framework and ours emphasize the comparison and evaluation of violations to each other and to the rewards.

While RL acquires policies through interaction with the environment and trial-and-error, planning relies on explicit models to compute optimal action sequences. *DARLING* integrates ASP-planning with RL to enable efficient and robust decision-making [LIS16]. The core idea is to use high-level planning to guide low-level RL, thereby combining the strengths of both paradigms. In DARLING, planning serves as a prior that steers the agent's behavior during early learning phases or when uncertainty is high. This integration allows the agent to explore more effectively and avoid suboptimal behaviors typically caused by sparse rewards or large state spaces. The planner is invoked selectively when the agent's policy lacks confidence or encounters unfamiliar situations. The system was evaluated in robotic navigation tasks where it showed improved learning speed and

policy robustness compared to pure RL. Similar to our approach, DARLING shows that the careful integration of planning and learning is essential for model-based reasoning, and the observed benefits align with our own findings regarding the balance between normative guidance and adaptive behavior.

The paper by Hatschka et al. (2023) [HCE23] utilizes ASP to represent SDL by formalizing deontic operators. This framework allows for the simulation of deontic paradoxes and of CTDs. Additionally, the authors employ weak constraints to assess violations of norms, a method that aligns closely with the approach in our own framework. The use of weak constraints offers a more flexible and nuanced evaluation of norm compliance, which is particularly valuable in environments where strict adherence to norms may not always be practical or desirable. This framework serves as a valuable reference for how ASP can be used to effectively model norms and highlights its potential for enhancing norm-guided behavior in reinforcement learning agents, ultimately facilitating the integration of more complex normative reasoning into agent-based systems.

Deontic Equilibrium Logic with eXplicit negation (*DELX*) is a formal framework that extends ASP for deontic reasoning by integrating the logic KD with both default and explicit negation [CCvdT23]. DELX builds on Deontic Equilibrium Logic (DEL), which itself extends equilibrium logic — a generalization of stable model semantics — to handle deontic modalities like obligation and permission. The key innovation in DELX is the ability to explicitly express the absence of facts while distinguishing it from classical default negation, which is essential for accurate normative modeling in scenarios involving contrary-to-duty (CTD) obligations or normative exceptions. The framework allows the encoding of deontic statements into logic programs in a way that supports reasoning under inconsistency and norm conflict. DELX theories can be compiled into deontic logic programs, making them executable under ASP solvers. It captures nuanced normative behavior such as soft constraints, defeasible norms, and hierarchical norm structures. This makes DELX particularly useful for simulating and analyzing normative systems where obligations may conflict, and compliance needs to be interpreted under ambiguity or partial observability.

Norm conflict resolution in stochastic environments can be addressed by combining Linear Temporal Logic (*LTL*) with probabilistic decision-making. Conflict Resolution Deterministic Rabin Automaton (*CRDRA*) deals with potential norm conflicts in non-deterministic environments [KS17]. This approach translates each norm into a deterministic automaton, and the agent's behavior is evaluated against these automata during policy optimization. Additionally, in situations where strict compliance is infeasible or undesirable norms can become suspended. Suspended norms are not discarded entirely but instead yield a penalty in the evaluation function, which decreases the importance of the norm. The resolution process involves computing an optimal policy that maximizes expected utility while balancing norm adherence and the cost of suspension. This allows the agent to act flexibly in uncertain and dynamically changing environments. The CRDRA-framework's combination of formal logic, probabilistic planning, and normative flexibility yields interesting configurations that could also be applied on the FrozenLake.

Moreover, LTL can be enriched with deontic operators, one example is deontic dynamic linear time temporal logic (*DDLTL*) [GMD13]. This framework utilizes ASP and extends traditional LTL by incorporating deontic operators to reason about normative concepts such as obligations, permissions, and prohibitions over time. DDLTL defines norms not only in terms of their temporal structure but also in relation to agent actions and contextual dynamics. Through its ASP-based implementation, the framework can compute so-called deontic answer sets, which represent consistent models of agent behavior that account for both temporal evolution and normative constraints. Moreover the framework can formally detect and classify norm violations. Specific logic formulas can be constructed to check both full compliance (hard compliance) and tolerated deviations (weak compliance). Weak compliance reflects practical considerations in normative reasoning, where enforcement is neither always possible nor desirable.

In [THM23], moral decision-making in multi-agent reinforcement learning is analyzed by utilizing philosophical theories of ethics into reward structures. The authors develop a framework in which agents have intrinsic reward functions defined by different normative ethical theories, such as *consequentialism* (outcomes-based reasoning), *deontology* (duty-based constraints), and *virtue ethics* (character-driven behavior). These moral models influence agent behavior independently of extrinsic environmental rewards and allow the embedding of value-driven policies that uphold general norms in social dilemmas. The framework supports heterogeneous moral agents operating in shared environments. This creates interactions that reflect real-world complexities, such as cooperation, moral disagreement, or exploitation. For instance, selfish agents which optimize only for personal gain may take advantage of agents following altruistic or duty-based policies. Additionally, the approach accounts for social outcomes by considering optimal outcome not only for the individual agent but also for others, which aligns with broader goals of fairness or collective welfare.

Pure logic programming can define norms with weights or orderings used to resolve potential conflicts. In the framework proposed by Berreby et al. [BBG17], norms are encoded with explicit conditions for their activation, deactivation, and violation, enabling both dynamic and temporal reasoning. Rather than treating norms as static constraints, the system models them as context-sensitive elements that evolve based on actions, events, and interactions between agents. Ethical evaluation is performed through a modular pipeline that integrates several complementary models, including a 'model of the good' (consequentialist evaluation), a 'model of the right' (duty-based reasoning), and a 'model of causality' (linking actions to outcomes). Alternative actions are evaluated in pairs and ranked according to how well they satisfy the active ethical principles. The framework omits the use of formal deontic operators, instead relying on explicit normative state transitions. The evaluation methods applied could be adapted to our environment to enrich policy interpretation and introduce more ethical post-evaluation mechanisms.

Furthermore, some norms in real-world applications are not always continuously active but can change their status. Considering the full lifecycle of norms is essential for realistic and context-sensitive normative reasoning. The framework proposed by Panagiotidi et al.

[PNV09] implements norm dynamics using ASP. In this framework, norms are represented as rules with temporal and contextual conditions that determine their lifecycle. A norm can be activated by specific events or conditions in the environment, remain active while its conditions hold, be deactivated once those conditions no longer apply, or be marked as violated if agents perform prohibited actions while the norm is active. The framework allows predictions when certain norms are likely to be triggered or rendered inactive, and adapt its strategy accordingly.

Although not directly applied in our experiments, the foundational work compiled in [AGNvdT13] provides a thorough account of norm representation, enforcement, and reasoning in symbolic agent systems. It establishes key conceptual and formal principles, including the use of deontic logic, sanctioning mechanisms, and institutional frameworks to govern agent behavior. While our approach integrates norms into learning-based agents through reinforcement learning, this work offers valuable background, emphasizing the importance of formal norm structures even in adaptive systems and supporting the case for hybrid models that combine logical reasoning with data-driven learning.

6.2 Norm Enforcing in RL

Action selection in reinforcement learning can be regulated by normative supervisory systems that evaluate the lawfulness of possible executions against a formalized body of norms. In [NBCG22], the authors implement such a supervisor using *SPINdle*, a theorem prover for defeasible deontic logic, to constrain the agent's behavior. The supervisor filters out norm-violating actions in each decision step, enforcing deontic constraints dynamically. If no fully compliant actions are available, the system utilizes a graded violation strategy, selecting the action with the lowest normative infraction based on a priority-based scoring function. This approach parallels the guardrail mechanism we employ, which similarly enforces norm constraints at runtime. However, both systems are lacking long-horizon planning. As a result, they are susceptible to suboptimal behavior in sequential settings, often converging to local optima that satisfy immediate constraints but neglect longer-term ethical objectives.

Deterministic action restrictions often fail to handle the uncertainty of non-deterministic environments. In contrast, *probabilistic shields* offer a more suitable approach by leveraging formal verification to act as runtime filters over agent actions [JKJ+20]. These shields are constructed from probabilistic models and specifications, ensuring that only actions satisfying safety constraints with a high probability are permitted. Unlike traditional hard constraints, the shields allow for some risk but quantify it probabilistically, balancing safety and performance. Compared to the Guardrail framework, which used the expected successor deterministically, probabilistic shields offer a more robust and formally grounded alternative ensuring compliance based on formal probabilistic guarantees.

An ASP-based decision-making framework can be utilized to retroactively integrate normative considerations into reinforcement learning policies, especially when such norms emerge after policy training. In this approach, a dedicated ASP component can be

invoked to generate emergency plans that resolve norm violations by providing compliant action alternatives in critical states [AE25]. This mechanism enables k-step planning in non-deterministic environments, allowing the agent to reason about norm-conforming paths under uncertainty. While the framework supports complete models, it requires several optimization strategies—such as rule pruning, incremental solving, and domain-specific heuristics—to maintain tractable execution times. In our work, we adopted this planning mechanism but with incomplete modeling to reduce complexity while still enabling normative interventions during runtime.

In norm-guided reinforcement learning (*NGRL*), non-compliance functions serve as punishments that guide agents toward morally desirable behavior by explicitly penalizing violations of predefined norms [Neu22]. This approach integrates normative constraints directly into the learning process through reward modification, enabling agents to balance instrumental goals with ethical considerations. Their framework enables norm penalties within multi-objective Q-learning (*MORL*), allowing for flexible trade-offs between normative adherence and task performance. While their implementation is tailored to moral scenarios, it overlaps conceptually with the reward-shaping mechanism used in our work, where extrinsic value signals are similarly adapted to influence agent behavior under complex social dynamics.

Restraining bolts serve as an external normative control mechanism for reinforcement learning agents by encoding specific prohibitions or obligations as constraints associated with punishment signals [NCT24]. Each bolt represents a distinct normative rule and assigns a numerical penalty to its violation, enabling the agent to learn behavior that minimizes cumulative punishment. This allows for the simultaneous enforcement of multiple norms and supports fine-grained control over agent compliance. Unlike traditional reward-shaping methods, restraining bolts work as an overlay architecture that can be modularly applied without altering the core learning objective. This makes them particularly suitable for dynamic or evolving norm systems. Compared to our framework, restraining bolts offer greater flexibility by allowing each norm to be independently defined, monitored, and activated as needed during training or deployment.

Zhao et al. [ZLA+18] integrated action norm penalties into a deep reinforcement learning framework to address fraud in e-commerce platforms. Their approach augments the standard reward function with norm-based penalty terms that discourage non-compliant actions, thereby guiding the policy towards legitimate and socially acceptable behaviors.

In reward shaping, the inclusion of domain-specific knowledge is essential for ensuring that the agent aligns with real-world constraints and expectations. In healthcare, where RL is increasingly applied to predict outcomes and recommend optimal treatments, the definition of what constitutes "optimal" is heavily context-dependent and often governed by ethical, legal, and normative frameworks. As discussed by Yu et al. [YLNY23], particularly in Section VII.B, multiple approaches have been proposed to integrate such constraints into the learning process. For example, preference-based reinforcement learning (*PRL*) can incorporate patient values or subjective well-being into decision-making, tailoring recommendations to individual preferences rather than relying solely

on clinical indicators. *MORL* allows for the optimization of multiple objectives, such as treatment efficacy, side-effect minimization, and compliance with medical regulations or ethical standards. Moreover, expert-in-the-loop methods introduce human oversight to further refine reward functions. Long-term reward considerations are also emphasized, as many medical decisions have delayed effects that may not be immediately observable but are crucial for evaluating treatment efficacy and safety. Unlike function-based formulations of rewards, these approaches acknowledge the subjective, uncertain, and context-sensitive nature of real-world environments. In such cases, strict guarantees of optimality may be less important than ensuring that the learned policies are interpretable, ethically sound, and aligned with stakeholder values.

Learned norms can be extracted from a deep RL policy. The *LEGIBLE* framework (Po**L**icy **E**valuation **G**u**I**ded **B**y ru**LE**s) is a method for integrating symbolic rule extraction and normative reasoning into the reinforcement learning process to improve policy performance and interpretability [TLTB25]. It operates in three main phases. First, it extracts behavioral rules from a trained deep reinforcement learning policy, identifying regularities and patterns in agent behavior that can be expressed as symbolic norms. Next, these rules are then generalized using external domain knowledge, enabling abstraction beyond individual state-action pairs and making the norms applicable across broader contexts. In the final phase, these generalized rules are used to guide further policy evaluation and improvement, allowing the system to reinforce compliant behaviors or correct violations through targeted policy updates. LEGIBLE does not merely shape rewards post hoc but introduces a loop where symbolic understanding of behavior feeds back into learning. This bridges the gap between black-box learning and explainable, norm-driven decision-making. The approach shows that structured rule guidance considering learned policies can yield policies that are both effective and interpretable. In contrast to architectures where norms are explicitly encoded from the beginning, LEGIBLE enables a data-driven pathway for discovering and refining norms, making it particularly useful in environments where such norms are not predefined but need to be inferred from behavior. While tabular Q-learning is typically more interpretable, integrating a rule-extraction mechanism like LEGIBLE could enhance our framework by analyzing and formalizing emerging normative patterns.

CHAPTER 7

Conclusion

In this work, we introduced a framework combining reinforcement learning, answer set programming and deontic reasoning to solve the FrozenLake-Environment of OpenAI gym under consideration of different norms.

We used several norms in our experiments which can be grouped into progress norms, simple norms, and contrary-to-duty (CTD) norms. They revolve around reaching the goal, both static and dynamic aspects of the level structure, and are faced with non-deterministic transitions of the environment. The progress norms and static norms were learned successfully by the agent in most setting, however, the dynamic norms and CTDs were harder to learn.

The experiments tested different application of norm-reasoning, namely within the agents planning model or as enforcing modules incorporated in the agent's interaction cycle. The results are mixed. For example, the internal planning-model accelerated learning and computes norm-adherent paths. However, it lacks awareness of the dynamic aspects, and the produced policies frequently failed to uphold the respective norms or to complete the task in more difficult levels. On the other hand, learning with full reward-shaping managed overall to learn all norms successfully, can be applied post-training to reprogram the moral of the agent, but without guided planning this approach might delude and mislead the agent into local maxima. Additionally, norm-based initialization strategies yielded also good results.

To conclude, the effectiveness of learning norms depends on their nature. Some are best acquired through planning with deontic reasoning, while others benefit from reward shaping and reinforcement learning. We anticipate that combining norm reasoning within the agent's model with a reward-shaping mechanism between the agent and the environment offers a promising approach for successfully learning diverse norms. Although our model was insufficient, the planning component remains highly beneficial and should not be discarded entirely.

As an outlook, future work should focus on refining the integration of norm reasoning within both planning and rewards. The limitations of the current model highlight the need for further investigation and the adoption of more sophisticated planning approaches. Reward shaping can be enhanced by utilizing different evaluation methods and improving state representation, ensuring that optimality guarantees are maintained for CTDs. Additionally, exploring combinations of the enforced strategies would provide deeper insights into their interplay and potential synergies.

Appendix

Overview of Generative AI Tools Used

Generative AI tools were used exclusively to assist with language refinement and grammar correction during the writing process. The tools used include ChatGPT, Deepseek, QuillBot, Writefull, and Grammarly.

No AI tools were employed in the design, implementation, or analysis of the research methodology.

List of Figures

Experiment Levels

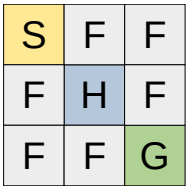

Figure A1: FrozenLake-Level: 3x3_A

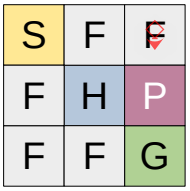

Figure A2: FrozenLake-Level: 3x3_B

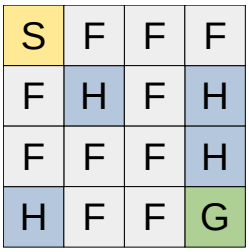

Figure A3: FrozenLake-Level: 4x4_A

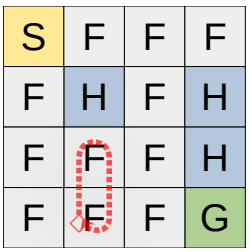

Figure A4: FrozenLake-Level: 4x4_B

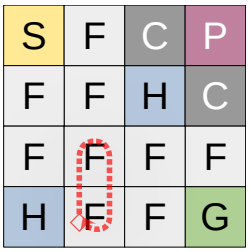

Figure A5: FrozenLake-Level: 4x4_C

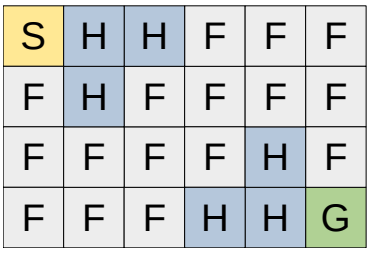

Figure A6: FrozenLake-Level: 6x4_A

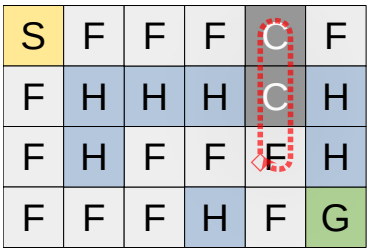

Figure A7: FrozenLake-Level: 6x4_B

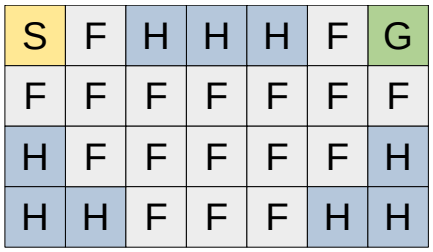

Figure A8: FrozenLake-Level: 7x4_A

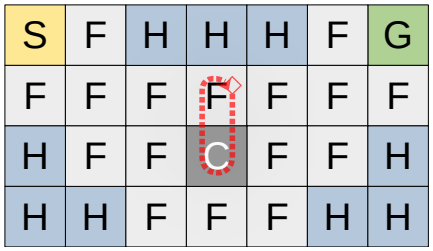

Figure A9: FrozenLake-Level: 7x4_B

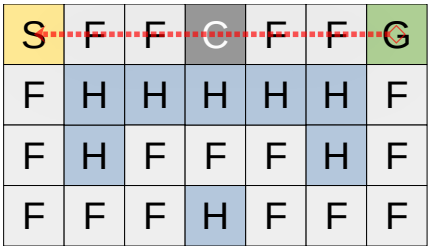

Figure A10: FrozenLake-Level: 7x4_C

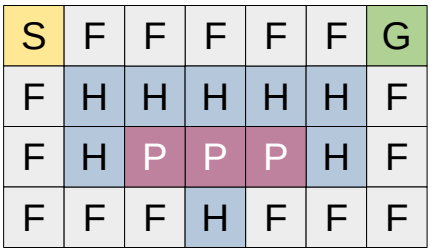

Figure A11: FrozenLake-Level: 7x4_D

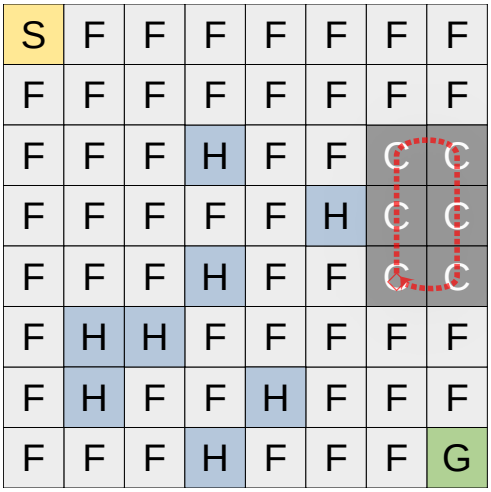

Figure A12: FrozenLake-Level: 8x8_A

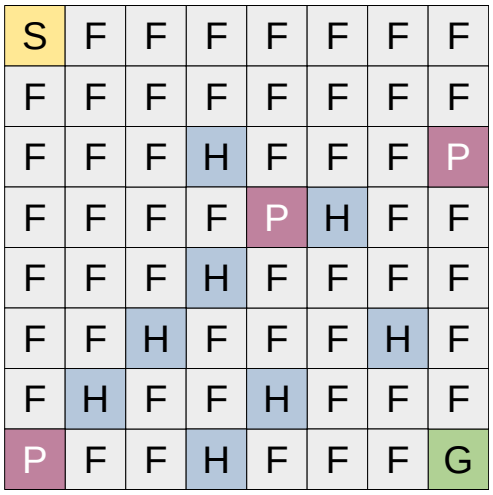

Figure A13: FrozenLake-Level: 8x8_B

List of Tables

List of Algorithms

Listings

Listing 7.1: deontic_norms_0.lp (Full set of norms)

```
1 % Full set of norms
2
3 % Norms:
4 % notReachedGoal
5 % occupiedTraverserTile
6 % turnedOnTraverserTile
```

```
% movedAwayFromGoal
% didNotMoveTowardsGoal
% leftSafeArea
% didNotReturnToSafeArea
% stolePresent
% missedPresents

#program always.

% The agent must not move away from the goal
forbidden(movedAwayFromGoal).
movedAwayFromGoal :- Y > X, 'distanceFromCurrentStateToGoal(X),
    distanceFromCurrentStateToGoal(Y).
currentViolation(forbidden(movedAwayFromGoal)) :- forbidden(
    movedAwayFromGoal), movedAwayFromGoal.
level(movedAwayFromGoal,1).

% The agent must move towards the goal
forbidden(didNotMoveTowardsGoal).
didNotMoveTowardsGoal :- Y >= X, 'distanceFromCurrentStateToGoal(X),
    distanceFromCurrentStateToGoal(Y).
didNotMoveTowardsGoal :- act(move(left)), currentState(L), leftEdge(L
    ).
didNotMoveTowardsGoal :- act(move(right)), currentState(L), rightEdge
    (L).
didNotMoveTowardsGoal :- act(move(up)), currentState(L), upperEdge(L)
    .
didNotMoveTowardsGoal :- act(move(down)), currentState(L), lowerEdge(
    L).
currentViolation(forbidden(didNotMoveTowardsGoal)) :- forbidden(
    didNotMoveTowardsGoal), didNotMoveTowardsGoal.
level(didNotMoveTowardsGoal,1).

% The agent must not be on the same tile as the traverser
forbidden(occupiedTraverserTile).
occupiedTraverserTile :- currentState(X), currentStateOfTraverser(Y),
     X=Y.
currentViolation(forbidden(occupiedTraverserTile)) :- forbidden(
    occupiedTraverserTile), occupiedTraverserTile.
level(occupiedTraverserTile,2).

% If the agent and traverser occupy the same tile, then the agent
    must move straight
forbidden(turnedOnTraverserTile).
turnedOnTraverserTile :- occupiedTraverserTile, act(move(X)),
    lastPerformedAction(move(Y)), X != Y.
currentViolation(forbidden(turnedOnTraverserTile)) :- forbidden(
    turnedOnTraverserTile), turnedOnTraverserTile.
level(turnedOnTraverserTile,3).
```

```
43
44 % The agent must not leave the safe area
45 forbidden(leftSafeArea).
46 leftSafeArea :- currentState(X), not safe(Y), tile(Y), X=Y, '
       currentState(U), safe(V), U=V.
47 currentViolation(forbidden(leftSafeArea)) :- forbidden(leftSafeArea),
       leftSafeArea.
48 level(leftSafeArea,2).
49
50 % If the agent left the safe area, it must return to a safe tile
51 forbidden(didNotReturnToSafeArea).
52 didNotReturnToSafeArea :- currentState(X), not safe(Y), tile(Y), X=Y,
       'currentState(U), not safe(V), tile(V), U=V.
53 currentViolation(forbidden(didNotReturnToSafeArea)) :- forbidden(
       didNotReturnToSafeArea), didNotReturnToSafeArea.
54 level(didNotReturnToSafeArea,3).
55
56 % The agent must not take any present
57 forbidden(stolePresent).
58 stolePresent :- takePresent(S).
59 currentViolation(forbidden(stolePresent)) :- forbidden(stolePresent),
       stolePresent.
60 level(stolePresent,3).
61
62 % The agent must take all available presents
63 forbidden(missedPresents).
64 level(missedPresents,3).
65
66 % The agent must reach the goal tile
67 obligatory(reachedGoal).
68 reachedGoal :- goalStateReached.
69 level(reachedGoal,4).
70
71 #program final.
72 currentViolation(obligatory(reachedGoal)) :- not reachedGoal.
73 currentViolation(forbidden(missedPresents(S))) :- forbidden(
       missedPresents), presentMissed(S).
74 level(missedPresents(S), L) :- level(missedPresents,L), presentMissed
       (S).
```

Bibliography

[ADBB17] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Process. Mag.*, 34(6):26–38, 2017.

[AE25] Sebastian Adam and Thomas Eiter. ASP-Driven Emergency Planning for Norm Violations in Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(14):14772–14780, Apr. 2025.

[AGNvdT13] Giulia Andrighetto, Guido Governatori, Pablo Noriega, and Leendert W. N. van der Torre, editors. *Normative Multi-Agent Systems*, volume 4 of *Dagstuhl Follow-Ups*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

[And58] Alan Ross Anderson. A reduction of deontic logic to alethic modal logic. *Mind*, pages 100–103, 1958.

[Åqv67] Lennart Åqvist. Good Samaritans, Contrary-to-Duty Imperatives, and Epistemic Obligations. *Noûs*, 1:361, 1967.

[BBG17] Fiona Berreby, Gauvain Bourgne, and Jean-Gabriel Ganascia. A Declarative Modular Framework for Representing and Applying Ethical Principles. In Kate Larson, Michael Winikoff, Sanmay Das, and Edmund H. Durfee, editors, *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, pages 96–104. ACM, 2017.

[BCP+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016.

[BET11] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.

[BF74] Morton B. Brown and Alan B. Forsythe. Robust Tests for the Equality of Variances. *Journal of the American Statistical Association*, 69:364–367, 1974.

[BK14] Christoph Beierle and Gabriele Kern-Isberner. *Methoden wissensbasierter Systeme - Grundlagen, Algorithmen, Anwendungen, 5. Auflage*. Computational intelligence. SpringerVieweg, 2014.

[Bl47] Welch Bl. The generalization of 'Student's' Problem when several different population varlances are involved. *Biometrika*, 34:28–35, 1947.

[CCvdT23] Pedro Cabalar, Agata Ciabattoni, and Leendert van der Torre. Deontic Equilibrium Logic with eXplicit Negation. In Sarah Alice Gaggl, Maria Vanina Martinez, and Magdalena Ortiz, editors, *Logics in Artificial Intelligence - 18th European Conference, JELIA 2023, Dresden, Germany, September 20-22, 2023, Proceedings*, volume 14281 of *Lecture Notes in Computer Science*, pages 498–514. Springer, 2023.

[Chi63] Roderick M. Chisholm. Contrary-to-duty imperatives and deontic logic. *Analysis*, 24:33–36, 1963.

[CKMS19] Pedro Cabalar, Roland Kaminski, Philip Morkisch, and Torsten Schaub. telingo = ASP + time. In Marcello Balduccini, Yuliya Lierler, and Stefan Woltran, editors, *Logic Programming and Nonmonotonic Reasoning - 15th International Conference, LPNMR 2019, Philadelphia, PA, USA, June 3-7, 2019, Proceedings*, volume 11481 of *Lecture Notes in Computer Science*, pages 256–269. Springer, 2019.

[DM15] Ernest Davis and Gary Marcus. Commonsense reasoning and commonsense knowledge in artificial intelligence. *Commun. ACM*, 58(9):92–103, 2015.

[For84] James William Forrester. Gentle Murder, or the Adverbial Samaritan. *The Journal of Philosophy*, 81:193–197, 1984.

[GHP+13] Dov M. Gabbay, John F. Horty, Xavier Parent, Ron van der Meyden, and Leendert van der Torre. *Handbook of deontic logic and normative systems*. 2013.

[GKKS19] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot ASP solving with clingo. *Theory Pract. Log. Program.*, 19(1):27–82, 2019.

[GKNS07] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. *clasp* : A conflict-driven answer set solver. In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings*, volume 4483 of *Lecture Notes in Computer Science*, pages 260–265. Springer, 2007.

[GL88] Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics for Logic Programming. In Robert A. Kowalski and Kenneth A. Bowen, editors,

Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes), pages 1070–1080. MIT Press, 1988.

[GL91] Michael Gelfond and Vladimir Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Gener. Comput.*, 9(3/4):365–386, 1991.

[GMD07] Davide Grossi, John-Jules Ch. Meyer, and Frank Dignum. On the Logic of Constitutive Rules. In Guido Boella, Leendert W. N. van der Torre, and Harko Verhagen, editors, *Normative Multi-agent Systems, 18.03. - 23.03.2007*, volume 07122 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

[GMD13] Laura Giordano, Alberto Martelli, and Daniele Theseider Dupré. Temporal deontic action logic for the verification of compliance to norms in ASP. In Enrico Francesconi and Bart Verheij, editors, *International Conference on Artificial Intelligence and Law, ICAIL '13, Rome, Italy, June 10-14, 2013*, pages 53–62. ACM, 2013.

[Gov18] Guido Governatori. Practical Normative Reasoning with Defeasible Deontic Logic. In Claudia d'Amato and Martin Theobald, editors, *Reasoning Web. Learning, Uncertainty, Streaming, and Scalability - 14th International Summer School 2018, Esch-sur-Alzette, Luxembourg, September 22-26, 2018, Tutorial Lectures*, volume 11078 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 2018.

[Gov24] Guido Governatori. An ASP implementation of defeasible deontic logic. *KI - Künstliche Intelligenz*, 2024.

[GST07] Martin Gebser, Torsten Schaub, and Sven Thiele. Gringo : A new grounder for answer set programming. In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings*, volume 4483 of *Lecture Notes in Computer Science*, pages 266–271. Springer, 2007.

[Han08] Jörg Hansen. The Paradoxes of Deontic Logic: Alive and Kicking. *Theoria*, 72:221–232, 2008.

[HCE23] Christian Hatschka, Agata Ciabattoni, and Thomas Eiter. Deontic Paradoxes in ASP with Weak Constraints. In *International Conference on Logic Programming*, 2023.

[JKJ+20] Nils Jansen, Bettina Könighofer, Sebastian Junges, Alex Serban, and Roderick Bloem. Safe Reinforcement Learning Using Probabilistic Shields

(Invited Paper). In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPIcs*, pages 3:1–3:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[KLM96] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement Learning: A Survey. *J. Artif. Intell. Res.*, 4:237–285, 1996.

[KS17] Daniel Kasenberg and Matthias Scheutz. Norm Conflict Resolution in Stochastic Domains. In *AAAI Conference on Artificial Intelligence*, 2017.

[Kub21] Miroslav Kubat. *An Introduction to Machine Learning, Third Edition.* Springer, 2021.

[Lif08] Vladimir Lifschitz. What Is Answer Set Programming? In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 1594–1597. AAAI Press, 2008.

[Lif19] Vladimir Lifschitz. *Answer Set Programming.* Springer, 2019.

[LIS16] 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.

[Mit97] Tom M. Mitchell. *Machine learning, International Edition.* McGraw-Hill Series in Computer Science. McGraw-Hill, 1997.

[MKS+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing Atari with Deep Reinforcement Learning. *CoRR*, abs/1312.5602, 2013.

[MKS+15] 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.

[MvdT00] David Makinson and Leendert van der Torre. Input/output logics. *Journal of Philosophical Logic*, 29:383–408, 2000.

[NBCG22] Emery A. Neufeld, Ezio Bartocci, Agata Ciabattoni, and Guido Governatori. Enforcing ethical goals over reinforcement-learning policies. *Ethics Inf. Technol.*, 24(4):43, 2022.

[NCT24] Emery A. Neufeld, Agata Ciabattoni, and Radu Florin Tulcan. Norm Compliance in Reinforcement Learning Agents via Restraining Bolts. In Jaromír Savelka, Jakub Harasta, Tereza Novotná, and Jakub Mísek, editors, *Legal Knowledge and Information Systems - JURIX 2024: The Thirty-seventh Annual Conference, Brno, Czech Republic, 11-13 December 2024*, volume 395 of *Frontiers in Artificial Intelligence and Applications*, pages 119–130. IOS Press, 2024.

[Neu22] Emery A. Neufeld. Reinforcement Learning Guided by Provable Normative Compliance. *CoRR*, abs/2203.16275, 2022.

[NHR99] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In Ivan Bratko and Saso Dzeroski, editors, *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999), Bled, Slovenia, June 27 - 30, 1999*, pages 278–287. Morgan Kaufmann, 1999.

[Nic02] Nicola Leone and Gerald Pfeifer and Wolfgang Faber and Thomas Eiter and Georg Gottlob and Simona Perri and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Log.*, 7:499–562, 2002.

[PNV09] Sofia Panagiotidi, Juan Carlos Nieves, and Javier Vázquez-Salceda. A Framework to Model Norm Dynamics in Answer Set Programming. In Matteo Baldoni, Cristina Baroglio, Jamal Bentahar, Guido Boella, Massimo Cossentino, Mehdi Dastani, Barbara Dunin-Keplicz, Giancarlo Fortino, Marie-Pierre Gleizes, João Leite, Viviana Mascardi, Julian A. Padget, Juan Pavón, Axel Polleres, Amal El Fallah Seghrouchni, Paolo Torroni, and Rineke Verbrugge, editors, *Proceedings of the Second Multi-Agent Logics, Languages, and Organisations Federated Workshops, Turin, Italy, September 7-10, 2009*, volume 494 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[Pri62] A. N. Prior. The Formalities of Omniscience. *Philosophy*, 37:114 – 129, 1962.

[PS96] Henry Prakken and Marek J. Sergot. Contrary-to-Duty Obligations. *Stud Logica*, 57(1):91–115, 1996.

[PvdT18] Xavier Parent and Leendert van der Torre. *Introduction to deontic logic and normative systems*. College Publications, 2018.

[Rei77] Raymond Reiter. On Closed World Data Bases. In Hervé Gallaire and Jack Minker, editors, *Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, France, 1977*, Advances in Data Base Theory, pages 55–76, New York, 1977. Plemum Press.

[Rei80] Raymond Reiter. A Logic for Default Reasoning. *Artif. Intell.*, 13(1-2):81–132, 1980.

[Ros44] Alf Ross. Imperatives and Logic. *Philosophy of Science*, 11:30 – 46, 1944.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998.

[Sch23] Bernhard Schiehl. Normative compliance in lexicographic multi-objective reinforcement learning agents. Master's thesis, Technische Universität Wien, 2023.

[TÅ91] James E. Tomberlin and Lennart Åqvist. Introduction to deontic logic and the theory of normative systems. *Noûs*, 25:109, 1991.

[THM23] Elizaveta Tennant, Stephen Hailes, and Mirco Musolesi. Modeling Moral Choices in Social Dilemmas with Multi-Agent Reinforcement Learning. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*, pages 317–325. ijcai.org, 2023.

[Tho93] Thomas Eiter and Georg Gottlob. Complexity Results for Disjunctive Logic Programming and Application to Nonmonotonic Logics. In *ILPS*, 1993.

[TLTB25] Martin Tappler, Ignacio D. Lopez-Miguel, Sebastian Tschiatschek, and Ezio Bartocci. Rule-Guided Reinforcement Learning Policy Evaluation and Improvement. *CoRR*, abs/2503.09270, 2025.

[Wie03] Eric Wiewiora. Potential-Based Shaping and Q-Value Initialization are Equivalent. *J. Artif. Intell. Res.*, 19:205–208, 2003.

[Yat34] F. Yates. Contingency Tables Involving Small Numbers and the $\chi 2$ Test, 1934.

[YLNY23] Chao Yu, Jiming Liu, Shamim Nemati, and Guosheng Yin. Reinforcement Learning in Healthcare: A Survey. *ACM Comput. Surv.*, 55(2):5:1–5:36, 2023.

[ZLA+18] Mengchen Zhao, Zhao Li, Bo An, Haifeng Lu, Yifan Yang, and Chen Chu. Impression Allocation for Combating Fraud in E-commerce Via Deep Reinforcement Learning with Action Norm Penalty. In *International Joint Conference on Artificial Intelligence*, 2018.