FACULTY OF ENGINEERING AND ARCHITECTURE

Learning Symbolic Representations from Demonstration for Interpretable Agent Behavior

Mattijs Baert

Doctoral dissertation submitted to obtain the academic degree of
Doctor of Information Engineering Technology

Supervisors
Prof. Pieter Simoens, PhD - Prof. Sam Leroux, PhD
Department of Information Technology
Faculty of Engineering and Architecture, Ghent University

July 2025

GHENT UNIVERSITY

ISBN 978-94-93464-03-2
NUR 985
Wettelijk depot: D/2025/10.500/63

Members of the Examination Board

Chair

Prof. Filip De Turck, PhD, Ghent University

Other members entitled to vote

Prof. Chris Develder, PhD, Ghent University
Prof. Ann Nowé, PhD, Vrije Universiteit Brussel
Prof. Alessandra Russo, PhD, Imperial College London, United Kingdom
Prof. Francis wyffels, PhD, Ghent University

Supervisors

Prof. Pieter Simoens, PhD, Ghent University
Prof. Sam Leroux, PhD, Ghent University

Dankwoord

Allereerst wil ik mijn promotoren, Pieter en Sam, oprecht bedanken voor de kans om dit PhD-avontuur aan te gaan. Het was een geweldige ervaring waarin ik veel heb geleerd, inspirerende mensen heb ontmoet en boeiende conferenties heb mogen bijwonen. Ik ben jullie enorm dankbaar voor jullie begeleiding gedurende dit traject. Ik kon altijd bij jullie terecht met mijn vragen, en het was telkens een plezier om nieuwe ideeën met jullie te delen. Jullie input heeft mijn werk voortdurend verrijkt, zonder jullie zou dit onderzoek simpelweg niet hetzelfde zijn geweest.

Ook een grote dank aan mijn mede-doctoraatstudenten van DECIDE en aan Casper voor de fijne sfeer op kantoor en de onmisbare snacks tijdens onze coffee breaks. Ondanks het risico op blessures, waren de teambuildings telkens iets om naar uit te kijken. Blij dat ik dit met jullie heb mogen meemaken. Keep pushing P!

Verder wil ik alle lesgevers in de opleiding industrieel ingenieur informatica bedanken. De passie en toewijding die jullie in jullie onderwijs steken zijn echt bewonderenswaardig. Ik zal deze opleiding blijven aanraden aan iedereen met een interesse in informatica.

Mijn diepe dank gaat uit naar Perle. Jij bent mijn veilige haven, mijn klankbord en steun in alles wat ik onderneem, dus ook in dit doctoraat. Toen ik zei dat ik naar een conferentie in Vancouver moest, had je sneller je koffers gepakt dan ik mijn presentatie af had. En terecht! Het was geweldig om zo'n avontuur samen te kunnen beleven. Ook Modest wil ik bedanken. Hij stelt dagelijks mijn geduld op de proef en eist veel van mijn energie op, maar maakt me vooral ontzettend gelukkig.

Ik ben mijn ouders en de rest van mijn naaste familie dankbaar voor hun onvoorwaardelijke steun en vertrouwen. Jullie liefde en geloof in mij hebben me geholpen om deze reis te maken. Daar ben ik jullie eeuwig dankbaar voor.

Tot slot wil ik mijn vrienden bedanken voor hun oprechte, en soms minder oprechte, interesse in mijn onderzoek. De vele mooie momenten in de scouts, op optredens, tijdens weekends of aan de toog van een Heulse kroeg zijn van onschatbare waarde voor mij. Merci voor alles.

Gent, 28 mei 2025
Mattijs Baert

Table of Contents

List of Figures

List of Tables

List of Acronyms

AI	Artificial Intelligence
ASP	Answer Set Programming
BC	Behavioral Cloning
CMDP	Constrained Markov Decision Process
CPO	Constrained Policy Optimization
CRL	Constrained Reinforcement Learning
CRM	Counterfactual Experiences for Reward Machines
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DFA	Deterministic Finite Automaton
DQN	Deep Q Learning
FOCOPS	First-Order Constrained Policy Optimization with Penalty
GAN	Generative Adversarial Network
GACL	Generative Adversarial Constraint Learning
GAIL	Generative Adversarial Imitation Learning
GPU	Graphics Processing Unit
ICRL	Inverse Constrained Reinforcement Learning
IG	Information Gain
ILASP	Inductive Learning of Answer Set Programs
ILP	Inductive Logic Programming
IRL	Inverse Reinforcement Learning
LCfD	Learning Constraints from Demonstrations
LC-QL	Logic Constrained Q-Learning
LfD	Learning from Demonstration
LLM	Large Language Model
LTL	Linear Temporal Logic
MaxEnt	Maximum Entropy
MCE-ICRL	Maximum Causal Entropy Inverse Constrained Reinforcement Learning
MCE-IRL	Maximum Causal Entropy Inverse Reinforcement Learning
ME-ICRL	Maximum Entropy Inverse Constrained Reinforce-

	ment Learning
ME-IRL	Maximum Entropy Inverse Reinforcement Learning
MDP	Markov Decision Process
MDPRM	Markov Decision Process with a Reward Machine
OCC	One Class Classification
OC-tree	One Class Tree
PCPO	Projected Constrained Policy Optimization
PDFA	Probabilistic Deterministic Finite Automaton
RGB	Red Green Blue
RGB-D	Red Green Blue Depth
RL	Reinforcement Learning
STL	Signal Temporal Logic
QL	Q-Learning
VLM	Vision Language Model
VME-ICRL	Variational Maximum Entropy Inverse Constrained Reinforcement Learning
XAI	Explainable Artificial Intelligence

Samenvatting
– Summary in Dutch –

Een autonome agent is een entiteit die zelfstandig beslissingen neemt op basis van zijn waarnemingen en doelstellingen. In de afgelopen jaren is het toepassingsgebied van zulke autonome agenten uitgebreid van afgebakende fabrieksterreinen naar omgevingen die gedeeld worden met mensen, zoals woningen, ziekenhuizen en stedelijke gebieden. In deze omgevingen wordt menselijk gedrag meestal geleid door een combinatie van expliciete regels (bijvoorbeeld verkeerswetten) en impliciete normen (zoals sociale etiquette). Een succesvolle integratie van autonome agenten vereist dat hun gedragsstrategieën aansluiten bij de waarden, regels en verwachtingen van menselijke gebruikers. Een zelfrijdende auto moet bijvoorbeeld niet alleen zijn passagiers efficiënt vervoeren, maar ook verkeersregels naleven, zoals stoppen voor verkeerslichten, voorrang geven aan voetgangers en rekening houden met de wegomstandigheden om de veiligheid te waarborgen. Het expliciet definiëren van dergelijke gedragsregels is echter niet triviaal omdat dit domeinkennis, inzicht in omgevingsrisico's en vaak ethische overwegingen vereist. Een slecht ontworpen doelstelling kan leiden tot agenten die strikt genomen aan de specificatie voldoen, maar niet het gewenste gedrag vertonen. Gezien deze uitdagingen richt dit proefschrift zich op het leren van symbolische representaties uit demonstraties, waarmee gedragsmatige patronen van agenten (zoals mensen) worden vastgelegd. Dit onderzoek slaat een brug tussen intuïtieve demonstraties (bijvoorbeeld observaties van menselijk gedrag) en formele, interpreteerbare modellen die kunstmatige agenten kunnen helpen bij het maken van beslissingen. Interpreteerbare representaties zijn niet alleen cruciaal voor het opbouwen van vertrouwen in AI-systemen, maar ook voor het *debuggen* en verbeteren ervan. Wanneer een systeem onverwacht gedrag vertoont, stelt een interpreteerbare representatie van de doelstellingen ontwikkelaars in staat om het besluitvormingsproces te analyseren, fouten te identificeren en het systeem te verfijnen. Afhankelijk van het type taak kunnen verschillende representaties worden gebruikt. In dit proefschrift richten we ons enerzijds op methodes voor het leren van beperkingen binnen het besluitvormingsproces. Anderzijds onderzoeken we hoe de structuur van een taak geleerd kan worden als

een aaneenschakeling van deeltaken. De voorgestelde methoden omvatten *Inverse Constraint Reinforcement Learning* (ICRL), neuro-symbolische AI, temporele logica en *Reinforcement Learning* (RL), en dragen bij aan schaalbare en interpreteerbare methodes voor leren uit demonstraties.

Het eerste deel van dit proefschrift richt zich op het leren van beperkingen op het gedrag van agenten. Dit is essentieel in omgevingen waarin agenten veiligheid moeten garanderen, ethische overwegingen in acht moeten nemen of binnen taak-specifieke grenzen moeten opereren. Een voorbeeld van zo'n toepassing is een zelfrijdende auto, waar beperkingen zoals verkeersregels en verkeersveiligheid moeten worden gerespecteerd om voorspelbaar en betrouwbaar gedrag te waarborgen. Het leren van beperkingen gegeven een doel en een reeks demonstraties staat bekend als *Inverse Constrained Reinforcement Learning* (ICRL). Binnen dit raamwerk wordt *Reinforcement Learning* (RL) toegepast tijdens of na het leerproces om een gedragsstrategie te leren die voldoet aan de afgeleide beperkingen. Een gangbare aanpak binnen ICRL is een iteratief proces van het afleiden van beperkingen en het bijwerken van de strategie. Tijdens het afleiden van beperkingen worden de meest waarschijnlijke beperkingen geïdentificeerd op basis van de demonstraties en een initiële strategie die geen beperkingen veronderstelt. Vervolgens wordt de strategie geüpdatet om rekening te houden met de geleerde beperkingen. Dit proces wordt herhaald tot de gedragsstrategie overeenkomt met de demonstraties. Traditionele ICRL-methoden zijn vaak beperkt tot discrete omgevingen of gaan uit van een deterministische transitiedynamiek, waardoor ze niet geschikt zijn voor complexe, realistische scenario's. Om deze beperkingen te overwinnen, stellen we een schaalbaar ICRL-raamwerk voor dat werkt in stochastische, continue omgevingen. We maken gebruik van het *maximum causal entropy*-principe en benaderen een kostenfunctie met een neuraal netwerk, waardoor beperkingen kunnen worden geleerd zonder enumeratie van alle toestanden of expliciete overgangsmodellen. We tonen de schaalbaarheid van deze aanpak aan met experimenten in gesimuleerde robot- en verkeersomgevingen. Hoewel deze methode schaalbaar is, gaat dit ten koste van de interpreteerbaarheid vanwege de black-box-aard van neurale netwerken. Om beperkingen op een interpreteerbare manier te leren, integreren we een *Inductive Logic Programming* (ILP)-engine in het proces. De ILP-engine generaliseert individuele beperkingen tot eerste-orde formules in predicatenlogica, wat het aantal iteraties dat nodig is om de ware set beperkingen af te leiden aanzienlijk vermindert. Deze methode verhoogt de interpreteerbaarheid, maar werkt onder specifieke aannames: het is vooral geschikt voor eenvoudige omgevingen met een discrete toestandsruimte, bekende overgangsdynamiek en predicaten gedefinieerd in de oorspronkelijke toestandsruimte. Om een schaalbare en interpreteerbare methode te ontwikkelen, herformuleren we het leren van beperkingen als een *one-class* classificatieprobleem. Onze aanpak maakt gebruik van *one-class* beslissingsbomen, waarbij elke split-

sing in de boom een ongelijkheid over toestandskenmerken weergeeft en zo een halfruimte in de oorspronkelijke toestandsruimte definieert. De resulterende beslissingsboom fungeert vervolgens als een interpreteerbare kostenfunctie. Uit experimenten in een gesimuleerde verkeersomgeving blijkt dat zowel expliciete regels (bijv. snelheidslimieten) als impliciete regels (bijv. een veilige afstand houden tot de auto ervoor) op een interpreteerbare manier kunnen worden geleerd. Met deze bijdragen draagt het eerste deel van dit proefschrift bij aan het vakgebied van *constraint learning* door schaalbaarheid, interpreteerbaarheid en toepasbaarheid in verschillende omgevingen in balans te brengen.

Het tweede deel van dit proefschrift richt zich op het leren van de structuur van een taak als een aaneenschakeling van deeltaken (i.e. een plan). Dit is vooral nuttig voor taken die bestaan uit meerdere discrete stappen, zoals het assembleren van een product of het bereiden van een maaltijd. We stellen methoden voor om subdoelen en hun onderlinge afhankelijkheden te identificeren, zodat complexe taken op een gestructureerde en interpreteerbare manier kunnen worden aangepakt. Allereerst presenteren we een methode voor het leren van temporele logica-formules rechtstreeks uit demonstraties. Temporele logica is een krachtige en expressieve taal voor het representeren van doelreeksen, gedeeltelijk geordende taken en veiligheidsbeperkingen in een interpreteerbaar formaat. De methode verfijnt iteratief een formule terwijl een RL-strategie opnieuw wordt getraind om aan de bijgewerkte taakspecificaties te voldoen. Hierdoor kunnen zeer interpreteerbare taakrepresentaties worden geleerd. Echter, omdat de formuleverfijning vereist dat alle kandidaat-formules en proposities worden geëvalueerd, brengt dit schaalbaarheidsuitdagingen met zich mee in complexere omgevingen. Als logische volgende stap onderzoeken we hoe een plan kan worden geleerd in complexe omgevingen met continue toestandsruimten en ongestructureerde demonstraties (zoals videobeelden). Aangezien subdoelen vaker voorkomen in demonstraties, identificeren we gebieden in de toestandsruimte die vaak voorkomen in de demonstraties met clusteringstechnieken. Het extraheren van subdoelen en hun temporele volgorde stelt ons in staat om een taakautomaat te construeren. Deze aanpak wordt gevalideerd met robot-gebaseerde objectmanipulatie taken, wat de toepasbaarheid in de echte wereld aantoont. Tot slot laten we de aanname van een vooraf gedefinieerde toestandsvector los en leren we subdoelen binnen een diepe neurale netwerk-feature ruimte, automatisch geëxtraheerd uit videobeelden. Vervolgens wordt RL gebruikt om een strategie te leren voor elk subdoel. Deze bijdragen bieden een raamwerk voor het leren van gestructureerde, interpreteerbare taakrepresentaties en slaan een brug tussen demonstraties en toepasbare, abstracte plannen, waarmee de weg wordt geëffend voor adaptieve en bekwame autonome agenten.

Summary

Over the last few years, the application domain of autonomous agents has expanded beyond caged factory environments to deployments in human-shared spaces, such as homes, hospitals, and urban settings. In these environments, human behavior is typically guided by a combination of explicit rules (e.g., traffic laws) and implicit norms (e.g., social etiquette). Successful integration of autonomous agents into these shared spaces requires their behavioral policies to align with the values, rules, and expectations of their human counterparts. For instance, a self-driving car must not only transport its passengers efficiently but also adhere to traffic regulations like obeying traffic lights, yielding to pedestrians, and accounting for road conditions to ensure safety. Explicitly defining such behavioral rules is nontrivial as it requires domain expertise, a deep understanding of environmental risks, and often, ethical considerations. Poorly designed objectives can lead to agent behavior that satisfies the literal specification of an objective but does not achieve the intended outcome. Given these challenges, this thesis aims to learn symbolic representations from demonstrations, encapsulating the behavioral patterns of agents (e.g., people). This research bridges the gap between intuitive demonstrations (e.g., observations of human behavior) and formalized, interpretable models that can guide artificial agents in decision-making. Interpretable representations are not only critical for fostering trust in AI systems but also for debugging and improving them. When a system behaves unexpectedly, an interpretable representation of the agent's objective allows developers to trace back the decision-making process, identify errors, and refine the system. Depending on the type of task, different representations can be used. In this dissertation, we focus on methods for learning constraints and high-level plans. The introduced methods span Inverse Constraint Reinforcement Learning (ICRL), neuro-symbolic AI, temporal logic, and reinforcement learning (RL), contributing to scalable and interpretable frameworks for learning from demonstration.

The first part of this dissertation focuses on learning constraints on agent behavior. This is crucial in environments where agents must ensure safety, adhere to ethical considerations, or operate within task-specific boundaries. An example of such a setting is autonomous driving, where constraints like traffic rules and road safety must be respected to ensure predictable and

reliable behavior. The task of inferring constraints from a reward function and a set of demonstrations is known as Inverse Constrained Reinforcement Learning (ICRL). In this framework, reinforcement learning (RL) is used during or after constraint learning to develop a behavioral policy that complies with the inferred constraints. A common approach to ICRL involves an iterative process of inferring constraints and updating a policy. During constraint inference the most likely constraints are identified based on expert demonstrations as well as a nominal policy that initially assumes no constraints. During the policy update, the nominal policy is refined to account for the constraints learned so far. This process is repeated until the nominal policy aligns with the expert demonstrations. Traditional ICRL methods are often limited to discrete settings or assume deterministic transition dynamics, making them impractical for complex, real-world scenarios where these conditions do not hold. To overcome these limitations, we propose a scalable ICRL framework designed for stochastic, continuous environments. We leverage the maximum causal entropy principle and approximate the cost function using a neural network, enabling constraint learning without the need for exhaustive state enumeration or explicit transition models. We demonstrate the scalability of this approach through experiments conducted in simulated robotic and traffic environments. However, while this method is scalable, it sacrifices interpretability due to the inherently black-box nature of neural networks. In order to learn constraints in an inherently interpretable way, we integrate an Inductive Logic Programming (ILP) engine into the process. The ILP engine generalizes individual constraints into logical formulas, which significantly reduces the number of iterations required to infer the true set of constraints. This method also enhances interpretability but operates under specific assumptions: it is best suited for simple environments characterized by a discrete state space, known transition dynamics, and predicates grounded in the original state representation. To develop a scalable and interpretable method, we reframe constraint learning as a one-class classification problem. Our approach leverages one-class decision trees, where each tree split represents an inequality over state features, effectively defining a half-space in the original feature space. The resulting decision tree functions as an interpretable cost function. Through experiments conducted in a simulation environment, we demonstrate that both explicit rules (e.g., speed limitations) and implicit rules (e.g., maintaining a safe distance from the car in front) can be effectively learned in an interpretable manner, even in complex environments. Through these contributions, the first part of this dissertation advances the field of constraint learning by balancing scalability, interpretability, and applicability to diverse environments.

The second part of the dissertation focuses on learning higher-level abstractions. This is useful for tasks consisting of multiple discrete steps such as assembling a product or preparing a meal. We propose methods for

identifying sub-goals and their interdependencies to address complex tasks in a structured and interpretable manner. We first present a method for learning temporal logic formulas directly from demonstrations. Temporal logic serves as a powerful and expressive language for representing goal sequences, partially ordered tasks, and safety constraints in an interpretable format. The method iteratively refines a temporal logic formula while retraining an RL policy to align with the updated task specifications. This process enables the learning of highly interpretable task representations. However, the formula refinement requires evaluating all candidate templates and propositions, which poses scalability challenges in more complex environments. As a logical next step, we examine how a high-level plan can be learned in complex environments with continuous state spaces and unstructured demonstrations (e.g., raw video data). Assuming that sub-goals appear more frequently in demonstrations, we identify high-density regions in the state space using clustering techniques. Extracting sub-goals and their temporal ordering allows us to construct a task automaton. This approach is validated on robotic object manipulation tasks, showcasing real-world applicability. Lastly, we relax the assumption of engineered feature representations consisting of object positions. Instead, sub-goals are inferred within a deep neural network feature space, automatically extracted from raw input data. RL is then used to train a policy for each sub-goal. These contributions collectively provide a framework for learning structured, interpretable task representations that bridge the gap between raw demonstrations and actionable, high-level plans, paving the way for more adaptive and capable autonomous agents.

1 Introduction

"When you change yourself, you change the world."

Gojira, "Silvera"

1.1 The past, present and future of AI

The origins of AI can be traced back to the mid-20th century, when early pioneers envisioned the possibility of creating machines that could mimic human cognitive functions, for instance, those that could pass the Turing test [1]. Among these visionaries was John McCarthy, who in 1956, during the seminal Dartmouth Conference, coined the term "Artificial Intelligence" [2]. The conference marked a pivotal moment as McCarthy, along with other pioneers like Alan Turing, Herbert Simon, and Marvin Minsky, laid the groundwork for AI as a formal academic discipline. The early decades of AI were characterized by rule-based systems, often referred to as symbolic AI. This approach was exemplified by expert systems, which used pre-defined rules and logic to mimic the decision-making processes of human experts in specific fields like medicine and law. Expert systems, such as MYCIN [3] and DENDRAL [4] in the 1970s, represented a significant leap in applying AI to real-world problems. However, they were constrained by their dependence on manually encoded knowledge and lacked the ability to learn from new data. By the early 2000s, AI underwent a transforma-

tive shift with the rise of machine learning, a subfield of AI that allowed computers to learn and make decisions based on data rather than relying on predefined rules. A crucial breakthrough came with the development of deep learning [5], a branch of machine learning that leverages neural networks with multiple layers to model complex patterns in data. Unlike symbolic systems, deep learning models learn complex representations from raw data, without manual feature engineering, unlocking unprecedented capabilities in fields such as image recognition, natural language processing, and autonomous systems. The transition from symbolic methods to deep learning was fueled by several factors: significant advancements in computational power, particularly through the use of Graphics Processing Units (GPUs), the availability of vast amounts of data, and innovations in neural network architectures. Milestones such as AlexNet [6], which dramatically improved state-of-the-art image classification, and AlphaGo [7], which defeated the world champion Go player Lee Sedol, exemplify the significant progress made in this domain. Furthermore, the development of large language models (LLMs) and vision language models (VLMs), such as ChatGPT [8] (LLM) and CLIP [9] (VLM). These models have significantly advanced the capabilities of AI by enabling sophisticated understanding and generation of natural language, as well as the ability to interpret and relate visual and textual information. This has unlocked a broad spectrum of applications, ranging from conversational agents and code generation to image captioning, visual search, and multimodal reasoning.

Despite impressive results, deep learning has faced criticism for its brittleness, as it is vulnerable to adversarial attacks [10]. Additionally, it has been criticized for its lack of explainability, as it lacks well-defined computational semantics or intuitive explanations, which raises concerns about the trustworthiness of AI systems [11]. Furthermore, deep learning has been noted for its lack of parsimony, as it requires excessive amounts of data and computational power during training, leading to unsustainable energy consumption [12]. One of the fundamental limitations of deep learning is that it is primarily a pattern recognition technique. As such, deep learning models lack the ability to reason, interpret abstract concepts, or understand causal relationships. These shortcomings have spurred the search for alternative or complementary approaches to address the gaps left by purely data-driven systems. To overcome the limitations of deep learning, researchers are increasingly turning to neural-symbolic systems, which combine the strengths of neural networks (for learning and pattern recognition) and symbolic AI (for reasoning and rule-based problem-solving) [13]. While neural networks excel at identifying patterns from large amounts of raw data, symbolic AI excels at representing explicit knowledge, handling abstract reasoning,

and understanding relationships between objects. By integrating these two approaches, neural-symbolic systems offer the potential to bridge the gap between perception and reasoning. For instance, in a neural-symbolic system, a neural network might process sensory data, such as images or text, while a symbolic component would interpret this data to make logical inferences or follow predefined rules. This hybrid approach offers both the adaptability of neural networks and the precision of symbolic reasoning, potentially leading to greater interpretability and robustness in decision-making. Some major AI breakthroughs are essentially neural-symbolic, even if not labeled as such. For instance, AlphaGo [7] combines Monte-Carlo tree search to simulate moves with neural networks to evaluate board positions. AlphaFold [14], which made a groundbreaking leap in predicting protein folding and earned global recognition, uses a hybrid of custom neural networks and classical symbolic tools. These examples illustrate the potential of neural-symbolic systems to achieve breakthroughs across domains by integrating perception with robust reasoning. Thus, the future of AI may be neural-symbolic, although major challenges remain in finding an effective interface between these systems.

1.2 Agents

Following the broader historical evolution of AI, a significant advancement in the field has been the development of AI agents. Unlike early AI systems, which often operated as isolated programs solving narrowly defined problems in controlled environments, agents represent a shift toward entities capable of perceiving, reasoning, and acting within complex, dynamic worlds. An agent is generally defined as a system situated in an environment that takes sensory inputs, processes them, and acts on the environment to achieve specific goals. This control loop enables agents to address real-world problems, such as navigating autonomous vehicles in traffic or performing robotic manipulation in unstructured human environments. However, developing AI agents comes with significant challenges. Agents must process noisy, high-dimensional sensory data and transform it into actionable representations. Agents operating in human spaces must align their behavior with both explicit and implicit human values to ensure harmonious integration. Furthermore, as agents assume increasingly complex roles, it is vital that their behavior remains understandable and predictable to foster trust and effective collaboration with humans. Many AI algorithms that excel in controlled environments struggle to scale to the complexity and stochasticity of real-world applications. These challenges highlight the need for robust frameworks that enable agents to operate reliably and intelligently in the real world.

1.3 Learning Strategies for Agents

Agent behavior can be described at different levels of abstraction. Consider, for example, a robotic arm. At the lowest level, control involves regulating the electrical currents sent to the actuators of its joints. At a higher level, the focus shifts to controlling the position of the end-effector. At an even higher level, the objective may be task completion, such as grasping an object. Objectives can be defined at each of these levels, and different learning strategies are suited for different levels of abstraction. In this section, we introduce various approaches for learning effective control policies, highlighting how some methods are better suited for fine-grained, low-level actions, while others excel at learning high-level task structures. The choice of technique depends largely on the specific problem at hand. Moreover, these methods can be combined allowing for a hierarchical and structured approach to learning agent behavior.

1.3.1 Reinforcement Learning

A widely adopted approach for training data-driven agents is Reinforcement Learning (RL) [15]. RL trains agents to make sequential decisions in an environment with the goal of maximizing a cumulative reward signal. At its core, RL relies on a feedback-driven process of trial and error: the agent experiments with different actions, observes their consequences, and adjusts its behavior based on rewards or penalties received. Over time, the agent refines its policy to optimize long-term rewards. This learning paradigm mirrors reinforcement mechanisms observed in biology, where reward signals influence behavior, decision-making, and survival strategies in animals.

RL methods span from foundational algorithms to advanced deep learning approaches. Early methods like value iteration and policy iteration involve iterating over all possible states and actions, enabling agents to learn optimal actions by calculating the expected rewards for each state-action pair. More advanced methods, such as Deep Reinforcement Learning (Deep RL), leverage neural networks to approximate value functions and policies. These approaches allow autonomous agents to operate effectively in complex, high-dimensional environments, such as robotics or video games, that are infeasible for traditional RL methods.

A significant challenge in RL is the reliance on a reward function to guide learning. Designing a reward function is a complex task that requires domain expertise, as it must clearly capture the desired outcomes for the agent. If the reward function is poorly designed, agents may exploit it by discovering shortcuts or behaviors that technically maximize rewards but

do so in unexpected and undesired ways, a phenomenon often referred to as reward hacking [16].

In real-world applications, agents often need to balance multiple and sometimes conflicting objectives to operate effectively. For example, an autonomous vehicle must not only transport passengers efficiently to their destination but must also comply with traffic laws, ensure passenger safety, and avoid accidents. This kind of multi-objective optimization is complex, as maximizing one goal (e.g., minimizing travel time) could compromise another (e.g., adhering to speed limits or maintaining safe distances from other vehicles). Constrained Reinforcement Learning (CRL) [17] is designed to address these complexities by training agents that can maximize rewards while respecting predefined constraints. In CRL, constraints are often represented by a cost function which returns a numerical cost value given the state of the environment and an action selected by the agent. The CRL learning objective consists of maximizing the cumulative reward while maintaining a cumulative cost below some threshold value. This way CRL embeds constraints into the learning process and ensures that agents make balanced decisions that consider both performance and critical real-world limitations. However, specifying constraints as a cost function (just as the reward function) is also a complex task which requires deep domain knowledge.

1.3.2 Learning from Demonstration

Humans often learn through trial and error, yet for many tasks, we also benefit from observing others. Learning from Demonstration (LfD) leverages this by allowing agents to learn tasks by imitating expert demonstrations. LfD can significantly improve sample efficiency since the agent requires less exploratory trial-and-error learning when provided with guided examples. Additionally, demonstrations are often easier to provide than designing a well-tuned reward function. A recent survey [18] categorizes LfD methods into three main types of outputs: policies, cost or reward functions, and plans. The choice of output depends on the level of abstraction best suited to the task at hand. For example, some tasks require learning precise, low-level actions. In contrast, other tasks may call for higher-level abstraction, where the agent learns the sequence of primitive actions, as well as their interdependencies, to accomplish complex tasks in a structured manner.

1.3.2.1 Learning Policies from Demonstration

This approach assumes that there exists a learnable function, or policy, that directly maps observations to actions. This approach is best suited when learning low-level controls with continuous actions. Behavioral Cloning

(BC) [19, 20] is a straightforward approach to policy learning that applies supervised learning directly to observation-action pairs from expert demonstrations. While simple and effective in some cases, policies learned through BC often struggle to generalize beyond the demonstrated behavior and may fail when states are encountered outside the training state distribution, leading to a notable decline in performance. A more sophisticated method in policy learning is Generative Adversarial Imitation Learning (GAIL) [21], which adapts the principles of Generative Adversarial Networks (GANs) to imitation learning. In GAIL, a generator network learns to replicate expert behavior by producing trajectories that are then evaluated by a discriminator. The discriminator's task is to distinguish between trajectories generated by the agent and those from the expert, while the generator iteratively improves to *mislead* the discriminator, refining its policy to match the expert's actions.

1.3.2.2 Learning Cost and Reward Functions from Demonstration

Methods in this category assume that optimal behavior is guided by an underlying, but unknown, cost or reward function. The goal of these methods is to infer this hidden function from expert demonstrations. These methods are also best suited for low-level control in continuous action spaces. Inverse Reinforcement Learning (IRL) is the problem of learning a reward function given demonstrations. Inverse Constrained Reinforcement Learning (ICRL) is the problem of learning a cost function given a reward function and demonstrations. Once the reward or cost function is learned, it can be used to train an agent using RL or CRL respectively.

Foundational IRL methods consider linear reward functions w.r.t. a feature representation of the state [22, 23]. In order to scale IRL to more complex problems neural networks are used to represent non-linear reward functions [24, 25]. IRL is considered an ill-posed problem because multiple reward functions are consistent with the observed expert demonstrations. To address this ambiguity, IRL methods incorporate additional optimization principles, such as maximum margin [26, 22, 23] or maximum entropy [27, 28, 29]. In maximum-margin IRL, the goal is to identify a reward function that maximizes the difference between the expert's policy and all other potential policies, effectively creating a *margin* that distinguishes optimal behavior. In contrast, maximum-entropy IRL seeks a policy that closely matches expert behavior while incorporating an entropy regularization term. By maximizing entropy, this method encourages the most uninformative distribution that remains consistent with the observed expert behavior. This approach minimizes assumptions about the expert's exact policy, making it particularly robust when demonstrations vary or

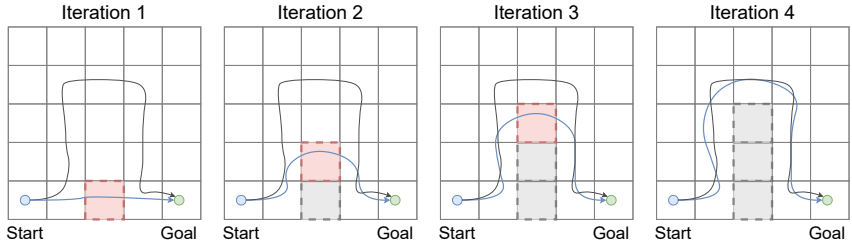


Figure 1.1: An illustrative example of the ICRL learning process in a gridworld environment, where the agent seeks the shortest path from the start state to the goal state. The expert's path is shown in black, while the learning agent's path is shown in blue. In each iteration, newly inferred constraint states are highlighted in red, and previously inferred constraint states appear in gray.

include inconsistencies. The entropy regularization allows the algorithm to model suboptimal demonstrator actions as random mistakes. An alternative framework, Bayesian IRL [30] extends beyond estimating a single reward function by inferring a full posterior distribution over possible reward functions. This method assigns probability mass to all reward functions that align with the observed demonstrations, provided they have support in the prior. However, Bayesian IRL faces significant scalability challenges. Consequently, its application has so far been limited to relatively simple environments, typically where the state and action spaces are manageable.

As mentioned earlier, desired behavior in human environments is highly constrained by explicit and implicit rules. ICRL focuses on inferring these constraints obeyed by expert agents, leveraging both interactions with the environment and demonstration data. In ICRL, the agent learns a representation of these constraints, which might take the form of a cost function, a set of states, or logical rules. This approach is especially useful when the primary goal can be easily specified through a reward function, yet underlying constraints are difficult to define explicitly. ICRL generally involves an iterative cycle, alternating between inferring constraints and updating the policy using CRL until the agent's policy sufficiently reproduces the expert's behavior. Figure 1.1 demonstrates this process in a gridworld environment where the agent aims to find the shortest path from a start state to a goal state given an expert trajectory. Initially, without any inferred constraints, the agent takes a direct path to the goal (Iteration 1). Noting that this path deviates from the expert's, the agent infers that a cell marked in red is likely infeasible, as the expert avoids it. The agent then updates its policy through CRL to reach the goal while respecting the inferred constraint. In Iteration 2,

the agent avoids the initial red region, only to observe that another area, also marked in red, is avoided by the expert. This alternating process of constraint inference and policy refinement continues until the agent's policy aligns closely with the expert's path, satisfying both the goal and implicit constraints.

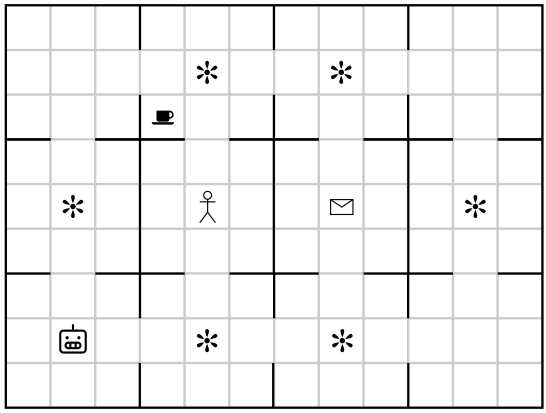
1.3.2.3 Learning Plans from Demonstration

The third category focuses on learning higher-level task structures. This type of tasks contain several subtasks that exhibit specific ordering constraints and interdependencies, e.g. assembling a device or preparing a meal.

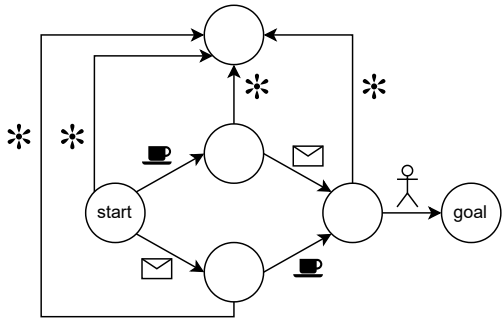
Although each task consists of a sequence of discrete subtasks or primitive actions, in most cases, demonstrations are provided in a continuous fashion. Because of this, segmentation plays a crucial role in task plan learning. Several methods have been explored for automated segmentation of demonstrations into subtasks, based either on the similarity of component subtasks [31, 32, 33] or on the occurrence of indicative events [34, 35].

Methods for learning plans from demonstrations can also be categorized by their learning output, i.e., which representation is learned to encode a plan. A first group of methods represent tasks as an automaton. In this case, the automaton's states represent the agent's progress within the task, while its edges denote the completion of sub-goals. Given an automaton, it is straightforward to obtain sequences of sub-tasks to fulfill the overall goal. However, when the number of states and edges increase, an automaton can become hard to interpret. From an automaton, trajectories in the low-level state space can be generated by using a library of movement primitives [36], RL [37, 38] or motion planning [39, 40]. Another group of methods considers learning task specifications using temporal logics. Temporal logics facilitate the specification of goal sequences, partial-order tasks, safety constraints, and various other conditions in a compact, interpretable manner. In practice, to use temporal logics for planning, an automaton that satisfies the logic formula should be synthesized [41]. However, the size of the synthesized automaton is worst-case double-exponential in the length of the formula which forms a bottleneck for the automaton synthesis process [42]. Alternatively, various method try to integrate logic specifications directly into an RL objective [43, 44, 45].

Consider an office environment where an agent (🤖) needs to collect the mail (✉), grab a mug of coffee (☕) and deliver this to an employee (🧍) while avoiding running into ornaments (✳). As illustrated in Figure 1.2, this task can be formalized using both an automaton and a temporal logic formula. The automaton represents plans as trajectories progressing from



(a)



(b)

$$G(\neg ✻) \wedge F(✉ \wedge XF(🧍)) \wedge F(☕ \wedge XF(🧍))$$

(c)

Figure 1.2: Visualization of an office gridworld environment [38] (a). Example of a task specification represented by an automaton (b) and a temporal logic formula (c).

the *start* state to the *goal* state, where each edge corresponds to the completion of a specific sub-goal. An additional terminal state is included to model a constraint: if the agent runs into an ornament, it transitions to this state, restraining the agent to reach the *goal* state. The temporal logic formula encodes the same objective using logical operators like negation ($\neg$), disjunction ($\vee$), and conjunction ($\wedge$), along with temporal operators such as *Globally* (G), indicating conditions that must always hold, *Finally* (F), indicating conditions that must eventually hold, and *Next* (X) indicating that a condition should become true at the next state.

1.3.3 Model-Based Learning Strategies

In contrast to RL and LfD, which primarily relies on interacting with the environment or observing expert behavior, model-based methods leverage explicit representations of the environment's dynamics to generate behavior. These strategies can be broadly categorized into two classes: those that rely on known system dynamics and those that involve learning world models from data.

Dynamics-based control methods operate directly on a known mathematical model of the system's dynamics. These models are often expressed as differential equations or discrete-time approximations that define how the system transitions between states in response to actions. Linear control methods, such as linear quadratic regulators [46], optimize control inputs by minimizing a quadratic cost function subject to linear system dynamics. Model Predictive Control (MPC) [47] is a powerful framework that generates behavior by solving a finite-horizon optimization problem at each time step.

For environments where the dynamics are unknown or too complex to model analytically, learned generative models can replace traditional dynamics equations. In Learning-Based MPC [48], a neural network or another machine learning model is trained to approximate the environment's dynamics. These learned models are then integrated into the MPC framework to predict future states and generate optimal actions. Model-based RL [49] combines aspects of RL with learned world models. In this paradigm, the agent learns a model of the environment's transition dynamics and reward function, enabling it to simulate future trajectories internally. By planning within this model, the agent can generate behavior with significantly fewer interactions with the environment.

Model-based methods offer several advantages, including sample efficiency, interpretability, and the ability to handle constraints explicitly. However, they face challenges in generalizing learned models to novel conditions,

ensuring robustness to model inaccuracies, and scaling to high-dimensional state-action spaces.

1.4 Research Contributions

The primary goal of my Ph.D. research was to develop algorithms that infer symbolic representations from demonstrations, encapsulating the behavioral patterns of agents (e.g., people). These interpretable "rules" bridge the gap between the actions observed and their underlying rationale, providing insights into "the why" behind behaviors. This capability is especially valuable in high-stakes applications such as autonomous systems. Interpretable representations are not only critical for fostering trust in AI systems but also for debugging and improving them. When a system behaves unexpectedly, an interpretable representation of the agent's objective allows developers to trace back the decision-making process, identify errors, and refine the system. Furthermore, they serve as a medium for collaborative interaction between human experts and AI. For instance, domain experts can directly inspect, validate, or amend the learned representations, enabling an iterative process of co-design that improves system performance and ensures alignment with real-world requirements. Finally, interpretability is crucial for addressing societal concerns surrounding the ethical use of AI and for lowering the barriers to its broader adoption. By prioritizing algorithms that produce interpretable representations, this research contributes to the development of neuro-symbolic systems in ways that humans can understand, trust, and control.

During my research, I focused on two levels of abstraction in agent behavior. In the first part of this dissertation, I concentrate on methods for learning constraints that govern low-level behavior. Constraints represent behavioral rules or limitations, such as safety requirements or task-specific restrictions. These are particularly important in environments where agents must ensure safety, adhere to ethical considerations, or operate within clearly defined boundaries. For example, in autonomous driving, constraints like traffic rules and road safety protocols must be learned and respected to ensure the agent's behavior is predictable, reliable, and safe. As such, constraints operate at the lowest level of abstraction, directly shaping the immediate actions an agent can take given an observation of its environment. In the second part of the dissertation, I shift focus to learning higher-level abstractions of behavior. These abstractions are especially useful for tasks that involve multiple discrete steps, such as assembling a product or preparing a meal. Here, I propose methods for identifying sub-goals and their interdependencies, providing a structured and interpretable framework for addressing complex tasks. Both parts of this research target distinct layers of behavioral

abstraction, and their applicability depends on the specific task at hand. In some cases, the two approaches can complement each other effectively. For instance, consider an autonomous pick-and-place robot operating in a warehouse: at the lower level, the robot must account for safety regulations, such as speed limits, maintaining safe distances, and yielding to humans. At the higher level, the robot must plan which items to pick and in what sequence to ensure optimal operation.

The overall objectives of this doctoral thesis are encapsulated in three key research questions. The first two research questions pertain to learning constraints (Part 1), while the third addresses learning plans (Part 2).

1. How can ICRL be extended to handle environments with stochastic transition dynamics in continuous domains?

Traditional ICRL methods are often limited to discrete environments or rely on strong assumptions about transition dynamics, limiting their applicability to real-world scenarios. This research explores how to scale ICRL to complex, real-world settings characterized by stochastic transitions and continuous state spaces.

To address these challenges, Chapter 2 introduces an objective grounded in the principle of maximum causal entropy, a variant of Shannon entropy that explicitly accounts for causal relationships between states and actions in trajectories. The cost function is parameterized by a neural network, enabling it to operate directly in the low-level state space and capture fine-grained constraints effectively. While this method demonstrates strong performance in complex environments, due to the black box nature of neural networks, the learned constraints are not interpretable.

2. How can symbolic representations of constraints be effectively learned from demonstrations?

Symbolic constraints offer interpretability and facilitate validation by humans. This question explores methods to infer these constraints directly from demonstrations, ensuring they generalize across diverse scenarios. The research examines how to bridge the gap between observed behavior and formal, human-readable representations.

In Chapter 3, we enhance the ICRL framework, depicted in Figure 1.1, by integrating a rule induction engine. After each iteration, the system attempts to infer first-order logic formulas (i.e., rules) that generalize the identified set of constraints. This approach offers two key advantages. First, by encapsulating multiple states within a single logical rule, it reduces the number of iterations required for constraint inference. Second, logic for-

mulas provide inherent interpretability, allowing for better insight into the learned constraints and facilitating human understanding and validation.

In Chapter 4, we approach the ICRL problem from a one-class classification perspective. We use one-class decision trees to effectively capture the state distribution of expert demonstrations. Each split in the tree corresponds to an inequality between a learned value and a single feature from the state space, defining a half-space. This decision tree is then employed as a cost function to train an agent using CRL. This method offers both interpretability and scalability to complex environments.

3. How can a symbolic representation of high-level plans be inferred from demonstrations?

High-level plans, such as task automata or temporal task specifications, are essential for guiding agents through complex tasks. This question focuses on extracting such plans from demonstration data.

In Chapter 5, we propose a methodology for learning task specifications in temporal logic directly from demonstrations. Our approach iteratively refines a temporal logic formula to better align with the observed demonstrations while simultaneously re-training a RL policy to adhere to the updated specification. This process bridges the gap between intuitive task demonstrations and formal task representations, enabling more accessible and accurate specification design.

The method introduced in Chapter 5 is constrained to environments with a discrete state space. In Chapter 6, we extend the scope to tackle complex environments with continuous state spaces by exploring how sub-goals and their temporal ordering can be extracted from unstructured demonstrations, such as raw video data. Our approach is based on the assumption that sub-goals are visited more frequently in demonstrations compared to other states. To identify these sub-goals, we leverage clustering techniques to detect high-density regions within the state space. This methodology is demonstrated on real-world robotic object manipulation tasks, showcasing its applicability to practical scenarios.

Building upon the work from Chapter 6, we relax the assumption of having an engineered feature representation that explicitly encodes all object positions. Instead, Chapter 7 proposes to infer sub-goals within a feature space automatically extracted by a deep neural network. We then demonstrate that with RL, we can effectively train a policy for achieving each sub-goal.

1.5 Publications

The research results obtained during this Ph.D. research have been published in scientific journals and presented at a series of international conferences and workshops. The following list provides an overview of these publications.

1.5.1 Journal Publications

[1] **M. Baert**, S. Leroux and P. Simoens, *Inverse Reinforcement Learning Through Logic Constraint Inference*. Published in Machine Learning, 112, 2023.

[2] **M. Baert**, P. Mazzaglia, S. Leroux and P. Simoens, *Maximum causal entropy inverse constrained reinforcement learning*. Accepted for Machine Learning.

1.5.2 Conference Publications

[1] **M. Baert**, S. Leroux and P. Simoens, *Intelligent frame selection as a privacy-friendlier alternative to face recognition* Presented at the AAAI workshop on Privacy-Preserving Artificial Intelligence (PPAI), virtual, 2021.

[2] **M. Baert**, S. Leroux and P. Simoens, *Learning logic constraints from demonstration* Presented at the international workshop on Neural-Symbolic Learning and Reasoning (NeSy), Siena, Italy, 2023.

[3] **M. Baert**, S. Leroux and P. Simoens, *Learning Safety Constraints From Demonstration Using One-Class Decision Trees* Presented at the AAAI workshop on Neuro-Symbolic Learning and Reasoning in the Era of Large Language Models (NucLeaR), Vancouver, Canada, 2024.

[4] **M. Baert**, S. Leroux and P. Simoens, *Learning Temporal Task Specifications From Demonstrations* Presented at the International Workshop on Explainable, Transparent Autonomous Agents and Multi-Agent Systems (EXTRAAMAS), Auckland, New Zealand, 2024.

[5] **M. Baert**, S. Leroux and P. Simoens, *Multi-stage task specification learning from demonstration* Presented at the RSS workshop on Task Specification for General-Purpose Intelligent Robots, Delft, Netherlands, 2024.

[6] **M. Baert**, S. Leroux and P. Simoens, *Learning Task Specifications from Demonstrations as Probabilistic Automata* Presented at the IEEE conference on robotics and automation (ICRA), Atlanta, USA, 2025.

[7] **M. Baert**, S. Leroux and P. Simoens, *Reward Machine Inference for Robotic Manipulation* Presented at the AAAI workshop on generalization in planning (GenPlan), Philadelphia, USA, 2025.

References

[1] A. M. Turing. *Computing machinery and intelligence*. Mind, 59, 1950.

[2] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon. *A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955*. AI magazine, 27(4):12–12, 2006.

[3] E. H. Shortliffe and B. G. Buchanan. *A model of inexact reasoning in medicine*. Mathematical biosciences, 23(3-4):351–379, 1975.

[4] R. K. Lindsay. *Applications of artificial intelligence for organic chemistry: the DENDRAL project*. (No Title), 1980.

[5] Y. LeCun, Y. Bengio, and G. Hinton. *Deep learning*. nature, 521(7553):436–444, 2015.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton. *Imagenet classification with deep convolutional neural networks*. Advances in neural information processing systems, 25, 2012.

[7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. *Mastering the game of Go with deep neural networks and tree search*. nature, 529(7587):484–489, 2016.

[8] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. *Improving language understanding by generative pre-training*. 2018.

[9] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. *Learning transferable visual models from natural language supervision*. In International conference on machine learning, pages 8748–8763. PmLR, 2021.

[10] C. Szegedy. *Intriguing properties of neural networks*. arXiv preprint arXiv:1312.6199, 2013.

[11] *Why Uber's self-driving car killed a pedestrian — economist.com*. https://www.economist.com/the-economist-explains/2018/05/29/why-ubers-self-driving-car-killed-a-pedestrian, 2018. [Accessed 23-01-2025].

[12] G. Marcus. *The next decade in AI: four steps towards robust artificial intelligence*. arXiv preprint arXiv:2002.06177, 2020.

[13] A. d. Garcez and L. C. Lamb. *Neurosymbolic AI: the 3rd wave. arXiv*. arXiv preprint arXiv:2012.05876, 10, 2020.

[14] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al. *Highly accurate protein structure prediction with AlphaFold*. nature, 596(7873):583–589, 2021.

[15] R. S. Sutton. *Reinforcement learning: An introduction*. A Bradford Book, 2018.

[16] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. *Concrete problems in AI safety*. arXiv preprint arXiv:1606.06565, 2016.

[17] E. Altman. *Constrained Markov decision processes: stochastic modeling*. Routledge, 1999.

[18] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard. *Recent advances in robot learning from demonstration*. Annual review of control, robotics, and autonomous systems, 3(1):297–330, 2020.

[19] M. Bain and C. Sammut. *A Framework for Behavioural Cloning*. In Machine Intelligence 15, pages 103–129, 1995.

[20] S. Ross, G. Gordon, and D. Bagnell. *A reduction of imitation learning and structured prediction to no-regret online learning*. In Proceedings of the fourteenth international conference on artificial intelligence and statistics, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[21] J. Ho and S. Ermon. *Generative adversarial imitation learning*. Advances in neural information processing systems, 29, 2016.

[22] P. Abbeel and A. Y. Ng. *Apprenticeship learning via inverse reinforcement learning*. In Proceedings of the twenty-first international conference on Machine learning, page 1, 2004.

[23] D. Silver, J. A. Bagnell, and A. Stentz. *Learning from demonstration for autonomous navigation in complex unstructured terrain*. The International Journal of Robotics Research, 29(12):1565–1592, 2010.

[24] M. Wulfmeier, P. Ondruska, and I. Posner. *Maximum entropy deep inverse reinforcement learning*. arXiv preprint arXiv:1507.04888, 2015.

[25] C. Finn, S. Levine, and P. Abbeel. *Guided cost learning: Deep inverse optimal control via policy optimization*. In International conference on machine learning, pages 49–58. PMLR, 2016.

[26] A. Y. Ng, S. J. Russell, et al. *Algorithms for inverse reinforcement learning*. In Icml, volume 1, page 2, 2000.

[27] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, et al. *Maximum entropy inverse reinforcement learning*. In Aaai, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[28] B. D. Ziebart, J. A. Bagnell, and A. K. Dey. *Modeling interaction via the principle of maximum causal entropy*. 2010.

[29] A. Boularias, J. Kober, and J. Peters. *Relative entropy inverse reinforcement learning*. In Proceedings of the fourteenth international conference on artificial intelligence and statistics, pages 182–189. JMLR Workshop and Conference Proceedings, 2011.

[30] D. Ramachandran and E. Amir. *Bayesian Inverse Reinforcement Learning*. In IJCAI, volume 7, pages 2586–2591, 2007.

[31] S. Niekum, S. Osentoski, G. Konidaris, and A. G. Barto. *Learning and generalization of complex tasks from unstructured demonstrations*. In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 5239–5246. IEEE, 2012.

[32] D. Kulić, C. Ott, D. Lee, J. Ishikawa, and Y. Nakamura. *Incremental learning of full body motion primitives and their sequencing through human motion observation*. The International Journal of Robotics Research, 31(3):330–345, 2012.

[33] F. Meier, E. Theodorou, F. Stulp, and S. Schaal. *Movement segmentation using a primitive library*. In 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 3407–3412. IEEE, 2011.

[34] Z. Su, O. Kroemer, G. E. Loeb, G. S. Sukhatme, and S. Schaal. *Learning manipulation graphs from demonstrations using multimodal sensory signals*. In 2018 IEEE international conference on robotics and automation (ICRA), pages 2758–2765. IEEE, 2018.

[35] A. Baisero, Y. Mollard, M. Lopes, M. Toussaint, and I. Lütkebohle. *Temporal segmentation of pair-wise interaction phases in sequential manipulation demonstrations*. In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 478–484. IEEE, 2015.

[36] S. Manschitz, J. Kober, M. Gienger, and J. Peters. *Learning to sequence movement primitives from demonstrations*. In 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 4414–4421. IEEE, 2014.

[37] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto. *Robot learning from demonstration by constructing skill trees*. The International Journal of Robotics Research, 31(3):360–375, 2012.

[38] R. T. Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith. *Reward machines: Exploiting reward function structure in reinforcement learning*. Journal of Artificial Intelligence Research, 73:173–208, 2022.

[39] S. Niekum, S. Osentoski, G. Konidaris, S. Chitta, B. Marthi, and A. G. Barto. *Learning grounded finite-state representations from unstructured demonstrations*. The International Journal of Robotics Research, 34(2):131–157, 2015.

[40] S. Pirk, K. Hausman, A. Toshev, and M. Khansari. *Modeling long-horizon tasks as sequential interaction landscapes*. arXiv preprint arXiv:2006.04843, 2020.

[41] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. *Temporal-logic-based reactive mission and motion planning*. IEEE transactions on robotics, 25(6):1370–1381, 2009.

[42] A. Camacho, J. Baier, C. Muise, and S. McIlraith. *Finite LTL synthesis as planning*. In Proceedings of the International Conference on Automated Planning and Scheduling, volume 28, pages 29–38, 2018.

[43] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta. *Q-learning for robust satisfaction of signal temporal logic specifications*. In 2016 IEEE 55th Conference on Decision and Control (CDC), pages 6565–6570. IEEE, 2016.

[44] X. Li, C.-I. Vasile, and C. Belta. *Reinforcement learning with temporal logic rewards*. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 3834–3839. IEEE, 2017.

[45] Z. Xiong, J. Eappen, A. H. Qureshi, and S. Jagannathan. *Constrained Hierarchical Deep Reinforcement Learning with Differentiable Formal Specifications*, 2022.

[46] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos. *The explicit linear quadratic regulator for constrained systems*. Automatica, 38(1):3–20, 2002.

[47] A. G. Richards. *Robust constrained model predictive control*. PhD thesis, Massachusetts Institute of Technology, 2005.

[48] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger. *Learning-based model predictive control: Toward safe learning in control*. Annual Review of Control, Robotics, and Autonomous Systems, 3(1):269–296, 2020.

[49] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. *Dream to control: Learning behaviors by latent imagination*. arXiv preprint arXiv:1912.01603, 2019.

Part I

Inverse Constrained Reinforcement Learning

2

Maximum Causal Entropy Inverse Constrained Reinforcement Learning

"Solitude is easy, you just do it on your own."

DIRK., "Milk"

Maximum Causal Entropy Inverse Constrained Reinforcement Learning

M. Baert • P. Mazzaglia • S. Leroux • P. Simoens.
Accepted for Machine Learning.

In the first part of this thesis, I explore how low-level constraints can be effectively learned from demonstrations, i.e., Inverse Constrained Reinforcement Learning (ICRL). This chapter specifically focuses on learning constraints represented by a cost function. Traditional ICRL methods are often restricted to discrete environments or rely on the assumption of deterministic transition dynamics. These requirements make the method impractical for more complex environments where such conditions are not met. In this chapter, we address these limitations by presenting a novel formulation of the ICRL problem. We prove convergence in a tabular setting and provide a practical implementation which scales to complex environments. We evaluate the effectiveness of the learned policy by assessing the reward received and the number of constraint violations, and we evaluate the learned cost function based on its transferability to other agents. Our method has been shown to outperform state-of-the-art approaches across a variety of tasks and environments, and it is able to handle problems with stochastic dynamics and a continuous state-action space. While this chapter focused on scalability by parameterizing the cost function with a neural network, the next chapter explores how constraints can be learned in a symbolic, interpretable representation.

2.1 Introduction

The behavior of individuals in today's society is shaped by social norms, which provide predictability, safety and efficiency in human interaction. Similarly, successful integration of artificial agents in the real-world requires *value alignment* of their behavioral policies [1, 2, 3]. As an example, a self-driving car should not only efficiently transport its passengers to their destination, but it should also adhere to traffic regulations, such as traffic signs, traffic lights, and road conditions, to ensure the safety of all road users. Furthermore, it is essential to program social norms into the agent prior to deployment to ensure that rule violations are avoided at all times. Learning value-aligned policies can be viewed as optimizing an objective subject to a set of constraints. In the framework of Reinforcement Learning (RL) this entails finding a policy that maximizes a reward function to which a weighted cost term was added that penalizes constraint violations. Not only does each combination of weights lead to a different Pareto optimal solution [4], also tuning them is a difficult and time-consuming process, as the task goal and the constraints often conflict with one another [5]. A more

effective alternative is the use of Constrained Reinforcement Learning (CRL) methods which adopt primal-dual methods on a constrained optimization problem, allowing to learn the optimal weights from data [6, 7]. Both RL and CRL methods require the manual specification of a cost function. Although it is straightforward to define constraints for simple environments, this is considerably harder for real-world settings, which often comprise multiple and implicit constraints. Returning to the traffic example, human drivers also adopt implicit rules, for example driving slower when traffic is heavy or leaving more space when driving behind a truck. To address this problem, we advocate for learning a cost function that encapsulates the environment's constraints from demonstrations provided by constraint-abiding agents, a paradigm known as Inverse Constrained Reinforcement Learning (ICRL).

Recent advancements in Inverse Reinforcement Learning (IRL) have facilitated the acquisition of reward functions from expert demonstrations in challenging environments [8, 9, 10]. However, there has been comparatively limited exploration in the realm of learning cost functions (i.e., ICRL). It is essential to note that in ICRL, the assumption is that the (goal-oriented) reward function of the expert is available, and the focus is solely on learning the constraints. Despite the apparent similarity between IRL and ICRL, a key distinction lies in how they handle states absent in expert demonstrations. The unvisited states, those not encountered by the expert, can be categorized into a group of constrained states and a group of states which are unconstrained but are associated with low reward potential. IRL fails to distinguish between these two groups of unvisited states, potentially leading to constraint violations if the agent explores states the expert did not visit. In contrast, ICRL explicitly separates these two groups by imposing high costs on constrained states, ensuring that the agent consistently avoids constrained states.

In this work, we propose a novel constrained optimization objective for addressing the ICRL problem. Unlike previous approaches [11, 12, 13, 14, 15], our methodology seamlessly scales to environments characterized by a continuous state-action space with unknown and stochastic transition dynamics. To avoid the potential issue of overfitting the constraints on the expert data, we adopt the maximum entropy principle [16]. This principle facilitates the learning of constraints that align with expert data while minimizing bias. In contrast to conventional methods [11, 12, 13, 14, 17, 18, 19], we adopt an alternative entropy definition that respects the causality inherent in states and actions within trajectories [20]. This distinction enables our method to be effectively applied to environments characterized by

stochastic transition dynamics.[1]

Our primary contributions encompass: i) the formulation of a novel ICRL objective grounded in the principle of maximum causal entropy, ii) the derivation of an exact solution for the proposed objective, supported by theoretical proofs, and iii) the development of an approximate method that extends the applicability of our approach to environments characterized by continuous state-action spaces. Our method simultaneously learns a cost function aligned with expert trajectories and a policy that maximizes the given reward function while minimizing the learned cost function. We evaluate the performance of the learned policy in terms of received rewards and the number of constraint violations across virtual and realistic environments. We explore the impact of varying levels of stochasticity in the environment's dynamics on these performance metrics. Additionally, we assess the transferability of the learned cost function to different types of agents with distinct reward functions. Finally, we conduct an ablation study to investigate the importance of different components within our proposed method. Empirical evaluation indicates that our method consistently outperforms state-of-the-art techniques across various benchmarks.

The structure of the chapter is as follows: In Section 2.2, we discuss related work. In Section 2.3, we present the necessary background information on constrained Markov decision processes and inverse reinforcement learning. Our proposed method is outlined in Section 2.4. We evaluate and compare the results of our method to existing state-of-the-art approaches in Section 2.5. Finally, we conclude and provide future directions in Section 2.6.

2.2 Related Work

2.2.1 Inverse Constrained Reinforcement Learning

A substantial body of current research aims to discern the set of constraints, or cost function, which maximizes the likelihood of the expert trajectories. This maximum likelihood objective can be solved by employing methodologies such as greedy algorithms [11, 12, 13, 14] or gradient-based methods [17, 18, 19, 21]. Kim et al. [22] formulate the ICRL objective as a zero-sum game between a policy player and a constraint player. Similarly to the previous methods, solving this objective yields a set of constraints corresponding to the states with high reward potential that were not visited by the expert. In contrast to our approach, these methods only consider environments with deterministic transition dynamics limiting their applicability in real-world

[1] Please refer to Sec. 2.7.1 in the appendix for a comprehensive explanation on why standard entropy is inappropriate in scenarios where stochastic transition dynamics are present.

scenarios.

An alternative line of work treats constraint learning as a one-class classification problem [23]. In this study, the central goal is to develop a model of expert behavior. Next, regions in the state-action space which are unlikely under the expert model are designated as constraints. In the same line of work, Lindner et al. [24] use a convex hull algorithm to obtain a safe set from the expert feature expectations. One major drawback of these methods is their vulnerability against overfitting on the expert data. In contrast, we adopt entropy regularization to learn an unbiased cost function. Papadimitriou et al. [15] offer a distinctive perspective by learning a distribution over potential constraints. Although this Bayesian approach introduces a nuanced understanding of uncertainty in the constraint inference process, it is limited to simple environments.

While some prior work [17, 25], considers soft constraints, the majority of methods concentrate on learning hard constraints [11, 12, 13, 14, 19, 22, 24]. In the context of reinforcement learning, hard constraints must always be satisfied, whereas soft constraints can be violated as long as they are satisfied on average (i.e., in expectation). In theory, our method supports both types of constraints. However, in this work, we focus on hard constraints, and, as of now, our experiments have exclusively involved hard constraints.

In contrast to our method, several approaches [11, 12, 13, 14, 15] necessitate the transition probability $p(s' \mid s, a)$ can be explicitly defined. Additionally, these approaches require multiple iterations over the entire state space. As a consequence, their applicability is confined to discrete environments characterized by a limited state space.

The constraint learning problem is inherently ill-defined, as many different sets of constraints can explain the same expert trajectories. In the extreme case, any state not visited by the expert might be considered a constraint [24]. To recover the smallest set of constraints which explain the expert trajectories, many methods [11, 12, 13, 14, 17, 18, 19] adopt the principle of maximum entropy [16]. All these methods, unlike ours, rely on the standard entropy definition, rendering them unsuitable for environments with stochastic transition dynamics. In contrast, the approach proposed by McPherson et al. [13] also employs maximum causal entropy, enabling constraint learning in stochastic environments. However, their method necessitates enumerating the entire constraint set, which makes it impractical not only for continuous state-action spaces but also for large discrete spaces. In a different approach, Kim et al. [22] leverage multitask demonstrations as a strategy to mitigate overfitting to expert demonstrations. However, this

strategy strengthens the dependence on the availability of diverse expert demonstrations and the existence of multiple tasks within the environment.

2.2.2 Learning From Experts

A variety of methods have leveraged human expertise to improve the performance of autonomous agents. Behavioral cloning relies on expert demonstrations to learn a policy through supervised learning [26, 27]. Despite its simplicity and efficiency in specific application contexts, it is possible there will be a mismatch between the training and testing state distribution, leading to a notable decline in performance. In preference learning, the goal is to learn a mapping between inputs and a ranking of those inputs which reflects the user's (or expert's) preferences [28, 29]. While humans often find it more straightforward to state preferences rather than providing complete demonstrations, the latter conveys a richer set of information. There are other works from the robotics community that focus on learning constraints from demonstrations. In [30], given the environment's dynamics, a set of unsafe trajectories is sampled, and constraints are learned by solving an integer programming problem using both the unsafe trajectories and the set of demonstrations. In [31], the scalability of this method is improved by exploiting constraint parameterization. Further, in [32], they extend their method to accommodate locally optimal demonstrations, and in [33, 34], they address how constraint uncertainty can be managed. However, their approach assumes that goal-satisfying trajectories can be easily sampled. This is a requirement that becomes impractical for tasks involving long control sequences within large or continuous action spaces. Most closely related to ICRL is IRL which has the goal of learning and subsequently optimizing a reward function [35, 8]. However, as we will demonstrate in the experiment section, IRL methods prove unsuitable for learning constraints.

2.3 Background

2.3.1 Constrained Markov Decision Process

A Markov decision process (MDP) is defined by a state space $\mathcal{S}$, an action space $\mathcal{A}$, a discount factor $\gamma \in [0, 1]$, a transition distribution $p(s' \mid s, a)$ which specifies the probability of transitioning to state s' when performing action a while in state s, an initial state distribution $\mathcal{I}(s)$ and a bounded reward function $R : \mathcal{S} \times \mathcal{A} \mapsto [r_{\min}, r_{\max}]$ specifying the scalar reward the agent receives for applying action a while in state s. We consider an agent interacting with the environment at discrete timesteps t generating a sequence of transitions called a trajectory $\tau = ((s_0, a_0, s_1), ..., (s_{T-1}, a_{T-1}, s_T))$ of length T. We define the reward of a trajectory as the sum of discounted rewards: $R(\tau) = \sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)$. At every timestep, the agent selects its

action based on a policy $\pi : \mathcal{S} \mapsto \mathcal{P}(\mathcal{A})$ which is a mapping from a state to a probability distribution over actions. The goal of forward reinforcement learning is to find the policy which maximizes the expected sum of discounted rewards: $\max_\pi \mathbb{E}_{\tau\sim\pi} R(\tau)$.

A constrained Markov decision process (CMDP) $\mathcal{M}^C$ [36] is defined as an MDP augmented with a non-negative bounded cost function $C : \mathcal{S} \times \mathcal{A} \mapsto [c_{\min}, c_{\max}]$ and a budget $\alpha \geq c_{\min}$. $C(s,a)$ denotes the cost of taking action a in state s and the cost of a trajectory is defined as the sum of discounted costs: $C(\tau) = \sum_{t=0}^{T-1} \gamma^t C(s_t, a_t)$. The goal of constrained reinforcement learning is defined as to find the policy which maximizes the expected sum of discounted rewards while the expected sum of discounted costs is smaller than the budget:

$$\max_\pi \mathbb{E}_{\tau\sim\pi} R(\tau) \;\text{ s.t. }\; \mathbb{E}_{\tau\sim\pi} C(\tau) < \alpha. \tag{2.1}$$

We set the lower bound of α to $c_{\min}$ since there exists no feasible solution for the CMDP when $\alpha < c_{\min}$. When $\alpha = c_{\min}$, constraints can be interpreted as hard constraints meaning the resulting policy incurs a minimum cost. When $\alpha > c_{\min}$, the resulting policy is allowed to obtain a cost $> c_{\min}$ in some cases, i.e., soft constraints.

2.3.2 Inverse Reinforcement Learning

Inverse Reinforcement Learning (IRL) is a learning problem which, given an expert data distribution $\mathcal{D}$ represented by a finite set of trajectories, tries to identify the reward function $R(s,a)$ the expert agent is optimizing. Assume the unknown reward function is defined as $R'(s,a) = \omega \cdot \phi(s,a)$ with $\omega \in \mathbb{R}^k$ a k-dimensional vector of real numbers and $\phi : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}^k_+$ denoting a fixed mapping from the state-action space to a k-dimensional vector of non-negative features. We define the feature representation of a trajectory as the sum of discounted feature vectors: $\phi(\tau) = \sum_{t=0}^{T-1} \gamma^t \phi(s_t, a_t)$. With forward RL we can obtain a policy which maximizes R' (i.e., the nominal policy). The problem of IRL can then be rephrased as finding values of ω such that the expected features when following the nominal policy $\mathbb{E}_{\tau\sim\pi}[\phi(\tau)]$ match the expected features under the expert data distribution $\mathbb{E}_{\tau\sim\mathcal{D}}[\phi(\tau)]$, i.e., feature expectation matching [35]. However, this is an ill-posed problem as many assignments of ω will result in matching expected features. To resolve this ambiguity, Ziebart et al. [20] proposed to choose the values ω that result in matching feature expectation and correspond to the nominal policy that maximizes the causal entropy, i.e., Maximum Causal Entropy Inverse Reinforcement Learning (MCE-IRL). Under Markovian dynamics the causal entropy of a trajectory is defined as the discounted sum of the conditional entropy of the current action given the current state:

$H(\tau) = \sum_{t=0}^{T-1} \gamma^t H(a_t \mid s_t)$. Although the original feature matching objective only considers reward functions that are defined as a linear combination of individual dimensions of ϕ, state-of-the-art methods are scalable to complex problems by parameterizing R' with a deep neural network [37, 8]. For an elaborate introduction to MCE-IRL we refer the reader to the work of Gleave et al. [38].

2.4 Maximum Causal Entropy Inverse Constrained Reinforcement Learning

In this section, we present Maximum Causal Entropy Inverse Constrained Reinforcement Learning (MCE-ICRL). First, we formalize our objective in Section 2.4.1. In Section 2.4.2, we present a detailed description of MCE-ICRL. To conclude, we provide a theoretical analysis in Section 2.4.3.

2.4.1 Problem Formulation

Inverse Constrained Reinforcement Learning (ICRL) methods aim to learn a cost function $C(s, a)$ that characterizes the constraints inherent in a specific environment. This is achieved by leveraging demonstrations $\mathcal{D}$ from agents that adhere to these constraints, and a known reward function $R(s, a)$ that defines their objectives. We will adopt the IRL terminology and will refer to constraint-abiding agents as expert agents. We define the following requirements for our novel ICRL method. Our method should (i) prevent overfitting on the expert data, (ii) scale to environments with a continuous state space and (iii) be applicable to scenarios with stochastic transition dynamics.

2.4.2 Algorithm

Although the goal of ICRL is to learn a cost function $C(s, a)$, we define the primal objective as identifying a stochastic policy $\pi(a \mid s)$, referred to as the nominal policy. As we will demonstrate in the following sections, solving this objective will enable us to also extract a cost function. We define the optimal nominal policy as the policy which maximizes the provided reward signal R and the causal entropy H, while concurrently aligning its feature expectations with the expert trajectories from $\mathcal{D}$:

$$\begin{aligned} &\max_{\pi \in \Pi} \quad \mathbb{E}_{\tau \sim \pi} \left[R(\tau) + \beta H(\tau) \right] \quad \text{s.t.} \\ &\mid \mathbb{E}_{\tau \sim \mathcal{D}} \left[\phi(\tau) \right] - \mathbb{E}_{\tau \sim \pi} \left[\phi(\tau) \right] \mid \leq \boldsymbol{\alpha}_k. \end{aligned} \tag{2.2}$$

With $\boldsymbol{\alpha}_k = (\alpha, \alpha, ..., \alpha) \in \mathbb{R}^k$ and where Π denotes the set of normalized policies with non-negative probabilities for all states and actions:

$$\pi \in \Pi \quad \Leftrightarrow \quad \pi(a \mid s) \geq 0 \ \text{ and } \int \pi(a \mid s) \, \mathrm{d}a = 1 \quad (\forall s \in \mathcal{S}). \tag{2.3}$$

We introduce the entropy coefficient β to strike a balance between the reward and entropy objectives. The alignment of feature expectations between the nominal policy and the expert data is enforced by imposing an inequality constraint with a specified budget α. To address the constrained optimization problem in eq. (2.2), we define the Lagrangian and determine its saddle points by solving the associated dual problem. The Lagrangian for the primal problem is obtained by replacing the feature matching constraint with a weighted penalty term in the objective, with weights $\lambda \in \mathbb{R}^k$ (i.e., the dual variable vector). Adhering to the Karush-Kuhn-Tucker (KKT) conditions, which stipulate that for any pair of optimal points (π, λ), all values of λ should be non-negative (Sec. 5.5.3 in [39]), the Lagrangian of the primal problem (eq. (2.2)) is expressed as:

$$\mathcal{L}(\pi, \lambda) = \mathbb{E}_{\tau\sim\pi}\left[R(\tau) + \beta H(\tau)\right] + \lambda \cdot \left(\mathbb{E}_{\tau\sim\mathcal{D}}\left[\phi(\tau)\right] - \mathbb{E}_{\tau\sim\pi}\left[\phi(\tau)\right] - \boldsymbol{\alpha}_k\right). \quad (2.4)$$

Then the dual problem can be defined as

$$\min_{\lambda>0} \max_{\pi\in\Pi} \mathcal{L}(\pi, \lambda). \quad (2.5)$$

The Lagrange multiplier functions to achieve equilibrium between maximizing reward and ensuring the agent's behavior aligns with that of the expert. In the MCE-ICRL framework, two sequential stages tackle the optimization process, alternating between refining the policy according to Equation (2.4) and adjusting the dual variables λ to optimize the same equation. Intuitively, it's logical to regard behaviors yielding high rewards but absent in the expert's trajectories as potential constraints. There likely exists a governing principle preventing the expert from executing this high-reward behavior. Consequently, when the nominal policy deviates significantly from the expert's, the values of λ will increase. This increase ensures that the penalties for deviating from these conceivable constraints become more pronounced, reinforcing alignment with the expert's behavior. As demonstrated in Section 2.4.3, solving the dual problem outlined in equation (2.5) not only enables us to learn a policy that adheres to the constraints of the environment but also allows us to derive a cost function $C(s, a) = \lambda \cdot \phi(s, a)$ which defines the constraints of the environment. Algorithm 2.1 provides an overview of the proposed approach. This algorithm fulfills all our predetermined requirements: (i) Incorporating entropy into our objective ensures that the learned policy remains unbiased. (ii) The gradient of the Lagrangian in Equation (2.4) can be computed w.r.t. the policy parameters, facilitating the use of highly scalable policy gradient reinforcement learning methods. (iii) By embracing causal entropy, we acknowledge the causal

relationships between states in an MDP, thus accommodating stochastic transition dynamics (see Appendix 2.7.1).

Algorithm 2.1: MCE-ICRL algorithm

Input: reward function $R(s,a)$, expert demonstrations $\mathcal{D}$
Parameter: number of iterations η

Initialize θ, λ and ζ
Learn nominal policy π_θ^0 with zero cost
Pre-train ϕ_ζ using $\tau \sim \pi_\theta^0$ and $\tau \sim \mathcal{D}$ (Sec. 2.4.3.2)
for $i \leftarrow 0$ ***to*** η **do**
  Learn π_θ maximizing eq. (2.4) w.r.t. to the policy parameters θ (Sec. 2.4.3.1, eq. (2.11)).
  Perform gradient descent on λ and ζ minimizing eq. (2.4) w.r.t. λ and ζ (Sec. 2.4.3.2, eq. (2.12) and eq. (2.13)).
end
Return cost function $C(s,a) = \lambda \cdot \phi(s,a)$, nominal policy π_θ

2.4.3 Algorithm Analysis

Theorem 1 *The dual problem in eq.* (2.5) *can be solved by alternately optimizing for π and λ. When the policy update is performed on a faster timescale than the updates on the dual variable vector, π and λ will converge to a local optimum. Proof. See chapter 6 of [40].*

Following Theorem 1, we iteratively solve the dual problem by optimizing the policy π (Sec. 2.4.3.1) and the dual variables λ (Sec. 2.4.3.2) alternately. Also in Sec. 2.4.3.2, we show we can replace the hard-coded feature representation ϕ with a neural network ϕ_ζ.

2.4.3.1 Policy Optimization

The maximization of the Lagrangian with respect to the policy corresponds to solving a planning problem, thereby rendering it possible to use RL techniques.

Proposition 2 *The optimal policy π^* maximizing eq.* (2.4) *is given by*

$$\pi^*(a_t \mid s_t) = \exp\left(\frac{1}{\beta}\big(Q^*(s_t,a_t) - V^*(s_t)\big)\right). \tag{2.6}$$

With the action-value function defined by

$$Q(s_t,a_t) = R(s_t,a_t) - \lambda \cdot \phi(s_t,a_t) + \gamma\, \mathbb{E}_p\left[V(s_{t+1})\right], \tag{2.7}$$

and the state-value function by

$$V(s_t) = \beta \log \int \exp\left(\frac{1}{\beta} Q(s_t, a)\right) \mathrm{d}a. \tag{2.8}$$

Proof. See Section 2.7.3.1 in the appendix.

Proposition 2 provides expressions for the optimal policy π^* (eq. (2.6)), the action value function Q (eq. (2.7)), and the state value function V (eq. (2.8)). The form of this action value function closely resembles the one found in the standard entropy-regularized RL objective [41]. It includes an additional term that subtracts $\lambda \cdot \phi(s, a)$ from the reward. This augmentation, originating from the feature matching objective as demonstrated in the appendix (Sec. 2.7.3.1), can be interpreted as a cost subtracted from the reward.

According to Theorem 3 below, the optimal policy can be derived through an iterative process involving policy evaluation and policy improvement. Policy evaluation entails inferring the action value function from any fixed policy, accomplished by iteratively applying a modified Bellman backup operator $\mathcal{T}^\pi$ given by

$$\mathcal{T}^\pi Q(s_t, a_t) \triangleq R(s_t, a_t) - \lambda \cdot \phi(s_t, a_t) + \gamma \, \mathbb{E}_p \left[V(s_{t+1}) \right], \tag{2.9}$$

with $V(s) = \mathbb{E}_\pi \left[Q(s, a) - \beta \log \pi(a \mid s) \right]$ (derived from eq. (2.6)). During the subsequent policy improvement step, the policy for each state is updated toward the optimal policy using the computed value functions:

$$\pi_{\text{new}} = \arg\min_{\pi' \in \Pi} \mathrm{D}_{\mathrm{KL}} \left(\pi'(\cdot \mid s_t) \,\middle\|\, \exp\left(\frac{1}{\beta}\big(Q(s_t, a_t) - V(s_t)\big)\right) \right). \tag{2.10}$$

Theorem 3 (Policy Iteration) *Continually applying policy evaluation (using eq. (2.9)) and policy improvement (using eq. (2.10)) starting from any $\pi \in \Pi$, converges to the optimal policy π^* for which $Q^{\pi^*}(s_t, a_t) \geq Q^\pi(s_t, a_t)$ for all $\pi \in \Pi$ and $(s_t, a_t) \in \mathcal{S} \times \mathcal{A}$, assuming $|\mathcal{A}| < \infty$. Proof. See Section 2.7.3.2 in the appendix.*

Policy iteration (theorem 3), however, is confined to a tabular setting with a limited state-action space. Recognizing this limitation, we introduce a practical algorithm designed to scale to continuous state-action spaces. In this context, we parameterize the policy using a neural network with parameters denoted as θ and adopt a policy gradient algorithm to attain the optimal policy.

Proposition 4 *Suppose that the policy π_θ is differentiable with respect to its*

parameters θ. Then

$$\nabla_\theta \mathcal{L}(\theta, \lambda) = \mathbb{E}_{\tau \sim \pi_\theta} \Bigg[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \Big(Q(s_t, a_t) - \beta \log \pi_\theta(a_t \mid s_t) - \sum_{t'=t}^{T-1} \gamma^{t'-t} \beta \Big) \Bigg]. \quad (2.11)$$

Proof. See Section 2.7.3.4 in the appendix.

The gradient in equation (2.11) closely resembles the standard policy gradient [42]. However, the maximum entropy objective introduces the additional term: $-\beta \log \pi_\theta(a_t \mid s_t) - \sum_{t'=t}^{T-1} \gamma^{t'-t} \beta$.

2.4.3.2 Optimization of Dual Variables

Proposition 5 *The Lagrange dual problem (eq.* (2.5)*) is a convex optimization problem. Proof. See Section 2.7.3.3 in the appendix.*

Because of proposition 5, we can use gradient descent to optimize the dual variables such that they converge to a global minimum. Then, theorem 1 and 3 and proposition 5 prove convergence for algorithm 2.1 in the tabular setting.

The step we need to take on the dual variable vector is obtained by taking the gradient of the Lagrangian with respect to λ:

$$\nabla_\lambda \mathcal{L}(\theta, \lambda) = \mathbb{E}_{\tau \sim \mathcal{D}} [\phi(\tau)] - \mathbb{E}_{\tau \sim \pi_\theta} [\phi(\tau)] - \boldsymbol{\alpha}_k. \quad (2.12)$$

Following each update, any negative values of λ are constrained to zero, as the KKT conditions necessitate that all values of λ should be non-negative. Feature expectations under the nominal policy are computed by iterating over a set of trajectories sampled from the learned policy. Conversely, for the feature expectations under the expert policy, the iteration involves the set of provided expert trajectories. In an intuitive sense, if we interpret $\lambda \cdot \phi(s, a)$ as a cost function $C(s, a)$ in equation (2.7), the update to λ is designed such that features occurring in trajectories sampled from the nominal policy, but not present in the expert data result in a higher cost. This feature matching enforces adherence to the expert data during the learning process.

Because the cost function is defined as the dot product of the transposed Lagrange multiplier λ and the feature representation $\phi(s, a)$, our method is limited to cost functions which are linear with respect to ϕ. In order to learn non-linear cost functions, we replace the hard-coded feature representation with a neural network with parameters ζ. We treat ζ as one of the dual

variables and thus update them together with λ. To do this we derive the gradient of the Lagrangian w.r.t. ζ:

$$\nabla_\zeta \mathcal{L}(\theta, \lambda, \zeta) = \lambda \cdot \left(\mathbb{E}_{\tau \sim \mathcal{D}} \left[\nabla_\zeta \phi_\zeta(\tau) \right] - \mathbb{E}_{\tau \sim \pi_\theta} \left[\nabla_\zeta \phi_\zeta(\tau) \right] \right). \tag{2.13}$$

The outputs of ϕ_ζ are passed through a sigmoid activation function to assure positive feature values. We obtained better results by pre-training the feature encoder network ϕ_ζ. To do this, we define a feature decoder network and train the encoder and decoder as an autoencoder trying to minimize the reconstruction error between the original data (i.e., encoder input) and the reconstruction (i.e., decoder output). The autoencoder is trained on nominal and expert data.

2.5 Experiments

We conducted a comprehensive validation of our approach across various settings, including a gridworld environment and the ICRL benchmark proposed by Liu et al. [19]. This benchmark encompasses five robotic domains and a realistic traffic environment. For our experiments, expert trajectories were sampled from a policy trained using reward-constrained policy optimization [7], adhering to ground truth constraints. Throughout the experiments, we optimize the policy using the approximate policy gradient method (Sec. 2.4.3.1). Specifically, the nominal policy was derived using the policy gradient method, Proximal Policy Optimization (PPO) [43]. Our set of validation experiments includes a study on the transferability of the learned cost function to other agents, an ablation study on the pre-training of the feature encoder, and an investigation into the influence of the hyperparameter β. Detailed information on hyperparameters and experimental settings can be found in Appendix 2.7.5. To assess the effectiveness of our method, denoted as MCE-ICRL, we compared it against three baseline methods:

- **Generative Adversarial Constraint Learning (GACL)**: This method is based on the well-established imitation learning method: generative adversarial imitation learning (GAIL) [8]. Following Malik et al. [18], GAIL can be adopted for ICRL by training the discriminator $D(s, a)$ to discriminate expert from nominal trajectories. The discriminator is trained such that $D(s, a)$ will output values close to 1 for state-action pairs drawn from expert trajectories and values close to zero for state-action pairs which only occur in the nominal trajectories (i.e., possible constrained state-action pairs). During the forward step, the policy is trained by maximizing $\bar{r}(s, a) = r(s, a) + \log D(s, a)$. Because $\bar{r}$ becomes $-\infty$ for constrained state-action pairs, the agent will try to satisfy all constraints.

- **Maximum Entropy Inverse Constrained Reinforcement Learning (ME-ICRL)** [18]: This method adopts the principle of maximum entropy [16] for modeling the likelihood of trajectories.
- **Variational Maximum Entropy Inverse Constrained Reinforcement Learning (VME-ICRL)** [19]: This method also builds on the principle of maximum entropy but models constraints as a beta distribution.
- **Learning Constraints from Demonstrations (LCfD)** [31]: This method uses hit-and-run sampling to generate unsafe (i.e., nominal) trajectories. By solving an integer program, a representation of the constraints can be learned.

We intentionally chose not to compare our method with approaches that assume knowledge of the transition function and require exhaustive iteration over the entire state-action space [11, 13, 12, 14]. Such a comparison would be unfair, as those methods assume complete knowledge of transition dynamics, whereas our approach only requires a simulator to sample trajectories based on a given policy.

We evaluate the nominal policy at different points during training by sampling trajectories from it and reporting the average obtained reward and the average constraint violation rate. The reward denotes the total reward an agent has received during a trajectory. The constraint violation rate is the fraction of all timesteps of a trajectory that violate at least one constraint. In all experiments, we consider hard constraints by setting the budget α to zero. This ensures that the feature expectations of the nominal and expert policy align (Eq.(2.2)).

2.5.1 Gridworld

We designed a gridworld environment where the agent's goal is to travel from a fixed initial location to a fixed goal location while avoiding constrained locations. A screenshot of this environment is shown in Sec. 2.7.4 in the appendix (Figure 2.5). Every step the agent receives a reward of -1 except when it reaches the goal state, the agent gets a reward of 1. Each episode lasts a maximum of 200 timesteps. We use this environment to examine the influence of stochastic environment dynamics on the performance of learning a cost function and retrieving the optimal policy. With a particular probability, the environment ignores the action taken by the agent and instead transitions it to a randomly chosen neighboring cell. We call this probability the stochasticity of the environment. Figure 2.1 shows the obtained reward and constraint violations at test time for increasing stochasticity. We observed a small decrease of reward and small increase of constraint violations for MCE-ICRL for increasing stochasticity. Non-causal

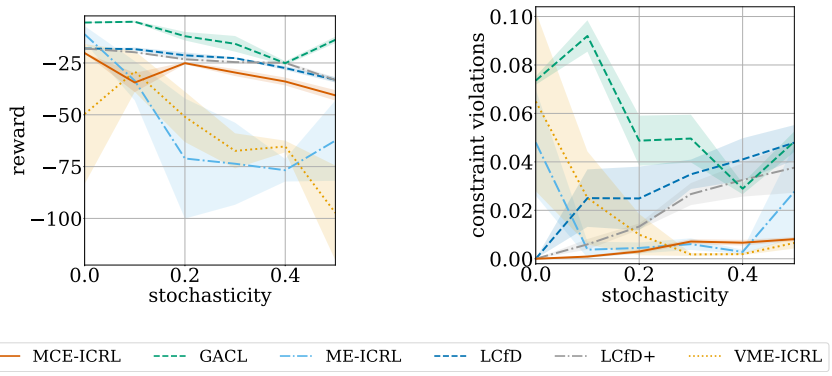
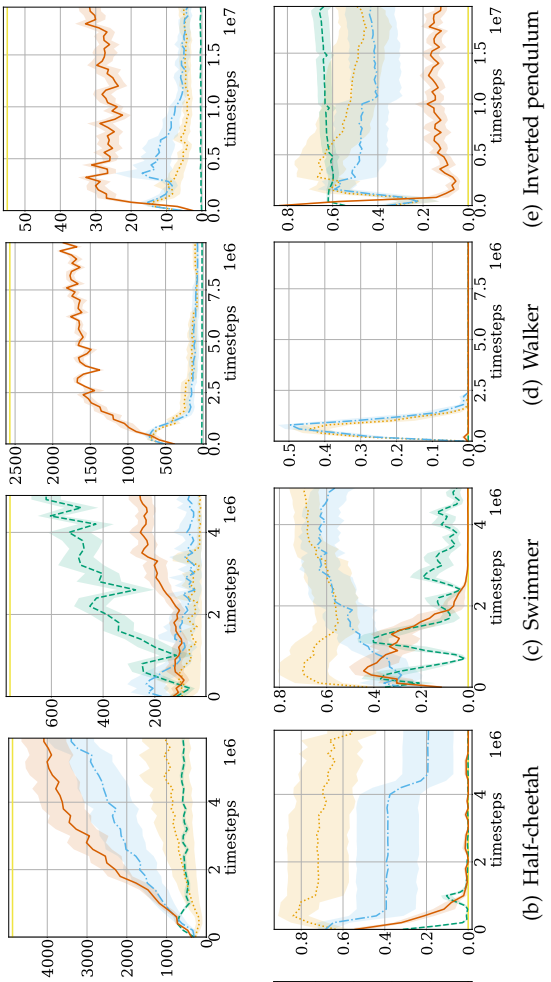


Figure 2.1: Evaluation of the different methods in the gridworld environment for increasing stochasticity: reward (left) and constraint violation rate (right) of trajectories sampled from the nominal policy after training. Results are averaged over 5 random seeds. The x-axis corresponds with the stochasticity. The shaded regions correspond to the standard error.

entropy approximates causal entropy for small stochasticity rates but fails when the randomness increases, this is reflected in the results of ME-ICRL and VME-ICRL. GACL consistently attains the highest rewards, albeit at the highest cost. The presence of scenarios wherein substantial constraint violations coincide with these elevated rewards raise concerns about the restrictiveness of the cost function. We conjecture that the discriminator may encounter challenges in generalizing the observed violated constraints in nominal trajectories to unseen state-action pairs. These results support our claim that IRL methods are not suitable for learning constraints. LCfD assumes that trajectories meeting start and goal constraints can be readily sampled. To narrow the sample space, a minimum reward threshold is set based on the average reward achieved in expert demonstrations. This approach works flawlessly in deterministic environments, consistently recovering the ground truth constraints. However, as stochasticity increases, LCfD fails to consistently recover the true constraints. To address stochastic dynamics, we introduce LCfD+, which lowers the reward threshold to include less optimal trajectories in the sampling process. This adjustment reduces the rate of constraint violations compared to LCfD. However, lowering the reward threshold expands the trajectory sampling space, leading to increased computational complexity, particularly when dynamics are unknown. Consequently, we find this method impractical for more complex tasks when dynamics are unknown. In Sec. 2.7.6.1 of the appendix, we report extended results of experiments in the gridworld environment.



(a) Ant (b) Half-cheetah (c) Swimmer (d) Walker (e) Inverted pendulum

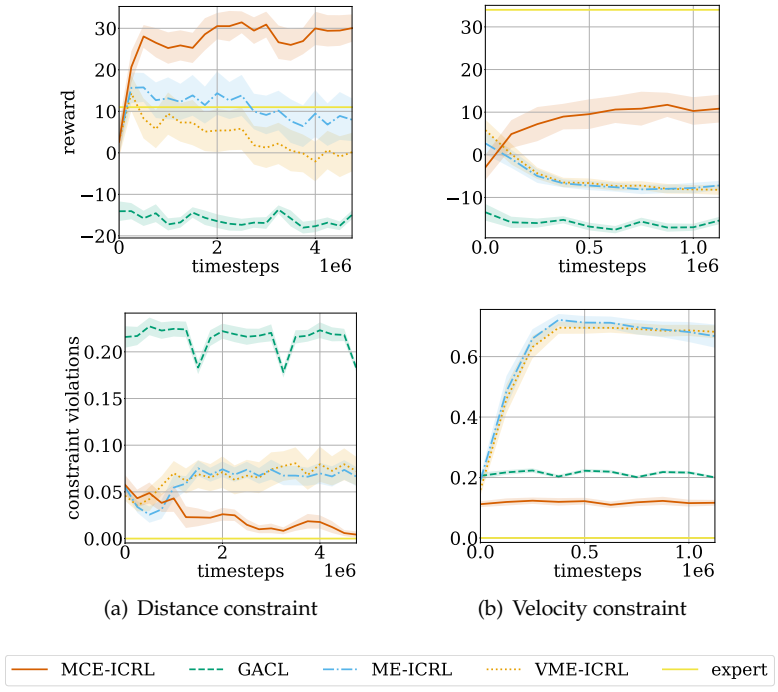
Figure 2.2: Evaluation of the different methods in the virtual robotics environments: reward (top) and constraint violation rate (bottom) of trajectories sampled from the nominal policy during training. Results are averaged over 10 random seeds. The x-axis is the number of timesteps taken in the environment during training. The shaded regions correspond with the standard error.

2.5.2 Virtual Robotics Environments

The virtual robotic environments are based on the MuJoCo environments from OpenAI Gym [44] but augmented with the constraints originally proposed by Malik et al. [18] and extended by Liu et al. [19]. We evaluate on five simulated robots: ant, half-cheetah, swimmer and inverted pendulum. Figure 2.6 (Sec. 2.7.4.2 in the appendix) depicts a screenshot of each environment. This environment is characterized by a continuous state-action space and deterministic dynamics. The reward of the ant agent is proportional to the distance traveled from the starting position. The reward of the half-cheetah, swimmer and walker agents is determined by the distance it moves each step. Because of their morphology, each of these agents can move "easier" in one direction than in all other directions, but the ground truth constraints invalidate moving in the "easy" direction. The agent controlling the inverted pendulum receives higher rewards when it is to the left of the starting position. However, the ground truth constraints correspond to these high reward locations. The agent should learn to balance the pendulum to the right from the origin receiving lower rewards but also lower costs. Results are depicted in Figure 2.2. In the ant environment MCE-ICRL performs comparable to the other methods. In the half-cheetah, walker and inverted pendulum domain our method results in higher rewards and less constraint violations. In the swimmer environment MCE-ICRL results in the lowest cost and higher rewards than ME-ICRL and VME-ICRL. GACL obtains higher rewards but with more constraint violations. For every task, we also plot the reward and constraint violation rate obtained during the expert trajectories. Note there is often still a gap between the performance of the learning agent and the expert.

2.5.3 Realistic Traffic Environment

The realistic traffic environment comprises a highway driving task. This task was proposed by Liu et al. [19]. This environment is constructed from the highD dataset [45] which is a dataset of naturalistic trajectories of vehicles on German highways. A scenario and an ego agent are randomly selected from the dataset for control and CommonRoad-RL [46] is used to get a state description of the ego car and its surroundings. This environment is characterized by a continuous state-action space and stochastic dynamics as the behavior of other agents in the environment is unpredictable and different expert agents act differently based on their preferences. The goal of the agent is to reach its destination at the end of the highway. Figure 2.7 (Sec. 2.7.4.3 in the appendix) visualizes the environment. The environment contains two scenarios, one with a minimum distance constraint between the ego car and other vehicles and one with a maximum velocity constraint.



(a) Distance constraint (b) Velocity constraint

Figure 2.3: Comparison of our method against various baselines in the realistic traffic environment, with distance constraints (left) and velocity constraints (right). The top two plots showcase the obtained rewards, while the bottom two plots depict the constraint violation rates of trajectories sampled from the nominal policy during training. The results are averaged over 10 random seeds, with the x-axis representing the number of timesteps taken in the environment during training. The shaded regions correspond to the standard error.

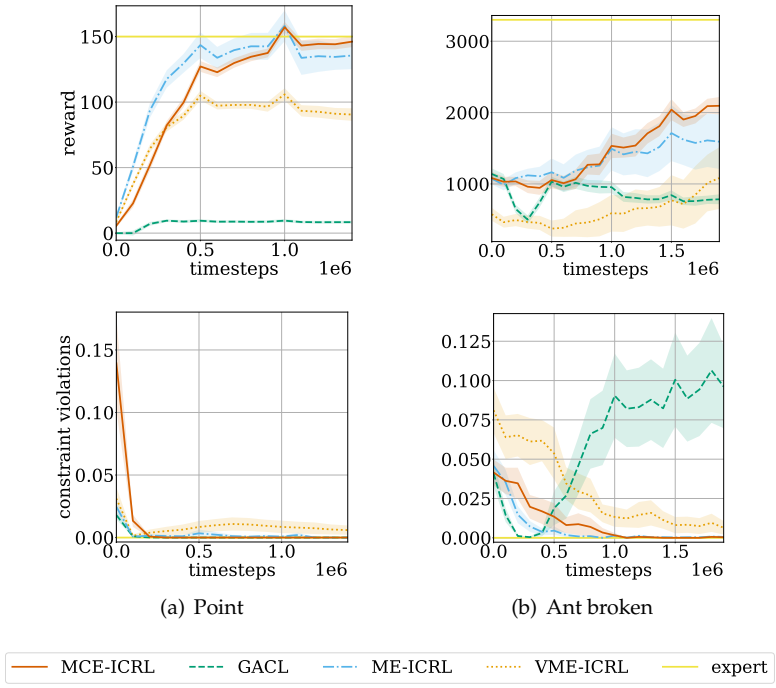


Figure 2.4: Comparison of our method against various baselines on the transferability of the cost acquired function. The cost function learned in the "ant" environment is transferred to the "point" agent (left) and the "ant broken" agent (right). The top two plots showcase the obtained rewards, while the bottom two plots depict the constraint violation rates of trajectories sampled from the nominal policy during training. The results are averaged over 10 random seeds, with the x-axis representing the number of timesteps taken in the environment during training. The shaded regions correspond to the standard error.

The results are depicted in Figure 2.3. Our method outperforms the other methods by a large margin which is not surprising as the traffic domain is characterized by high stochasticity. The ground truth constraints were arbitrarily chosen by Liu et al. [19] and are in some cases violated in the starting state of the ego agent. Because of this, no method, even not the expert agent trained given the ground truth constraints, is able to reduce the cost to zero.

2.5.4 Transfer Learning

When a cost function is learned for a specific agent, it could be beneficial when that cost function can be transferred to other types of agents operating

in the same environment. This would prevent us from learning a cost function for every kind of agent. Note that constraints are often environment specific and apply to different types of agents, e.g., traffic rules apply to all vehicles on the road. To this end, we assess the transferability of a learned cost function to other types of agents which possibly have other morphologies and reward functions. We perform the transfer learning experiments proposed by Malik et al. [18]. The cost function learned for the "ant" agent is transferred to a "broken ant" agent for which two of its legs are disabled. We also transfer this cost function to the point agent from OpenAI Gym [44]. The reward function of the point robot encourages the agent to move in counterclockwise in a circle at a distance of 10 units from its initial location. We adopt reward constrained policy optimization [7] to train agents on the given reward function and the transferred cost function. The results are depicted in Figure 2.4. For both agents our method performs slightly better than ME-ICRL and outperforms VME-ICRL and GACL by a large margin in both reward and number of constraint violations.

2.5.5 Ablation Studies

2.5.5.1 Influence of the Hyperparameter β

β determines the weight assigned to the entropy term in the objective function and can be interpreted as a regularization coefficient to enforce randomness into the learned policy π. Higher values of β lead to policies exhibiting reduced bias toward paths with identical cumulative rewards. Consequently, the associated cost function will invalidate a minimal set of state-action pairs required to match feature expectations of the nominal policy with those of the expert. Table 2.1 reports the reward and the constraint violation rate of trajectories sampled from a policy learned using MCE-ICRL during testing for the virtual robotic environments. We can conclude that the value of β heavily influences the received reward and the number of constraint violations. If β is too small, our method overfits on the expert trajectories leading to small rewards and possibly more constraint violations. For high values of β, the nominal policy becomes more random which also results in lower rewards and more constraint violations. For each environment, the value of β should be tuned to obtain optimal results.

2.5.5.2 Pre-Training the Feature Encoder

To assess the importance of pre-training the feature encoder, we train a version of MCE-ICRL without pre-training. Table 2.1 reports the received reward and constraint violation rate of the nominal policy at test time for the virtual robotic environments. From these results it is clear that pre-training the feature encoder provides a significant increase of the received reward.

Table 2.1: Ablation study on the pre-training of the feature encoder and the entropy coefficient β, the table contains for every agent the received reward and the constraint violation rate at test time $\pm$ standard error, averaged over 10 random seeds and 10 trajectories per seed.

		Ant	Half-cheetah	Swimmer	Walker	Inverted pendulum
Reward	$\beta = 1e{-}5$	5985±259	3895±295	177±52	1647±100	27±3
	$\beta = 1e{-}4$	**7338±211**	**4085±565**	**257±44**	**1858±44**	26±3
	$\beta = 1e{-}3$	6057±313	3499±492	253±54	1746±81	**32±3**
	$\beta = 1e{-}2$	6228±316	2377±631	192±43	1732±77	29±4
	no pre-training	7131±313	3609±490	165±59	1656±71	21±1
Constraint violation rate	$\beta = 1e{-}5$	**0±0**	**0±0**	0.33±0.09	**0±0**	0.22±0.05
	$\beta = 1e{-}4$	**0±0**	**0±0**	**0.002±0.001**	**0±0**	**0.11±0.03**
	$\beta = 1e{-}3$	**0±0**	0.002±0.002	0.12±0.04	**0±0**	.12±0.03
	$\beta = 1e{-}2$	**0±0**	0.024±0.012	0.11±0.05	**0±0**	0.17±0.02
	no pre-training	**0±0**	**0±0**	0.22±0.09	**0±0**	0.23±0.04

Section 2.7.6 in the appendix provides additional results on the effect of pre-training the feature encoder and β during training and for the other environments.

2.6 Conclusion and Future Work

We introduced Maximum Causal Entropy Inverse Constrained Reinforcement Learning (MCE-ICRL), a novel ICRL method capable of learning constraints in environments with unknown stochastic dynamics. Our contribution includes a theoretical proof of convergence in a tabular setting, demonstrating the foundational robustness of our approach. Additionally, we presented an approximation allowing our method to effectively scale to complex problems featuring continuous state-action spaces. Our empirical results showcase the superior performance of our method compared to state-of-the-art approaches. Notably, the learned cost functions exhibit successful transferability to diverse agents with distinct reward functions. Despite these achievements, our method does not match expert agent performance in certain scenarios, indicating areas for potential improvement. An ablation study underscores the significance of selecting an appropriately tuned value for the parameter β, suggesting that an automatic tuning mechanism, as proposed by Haarnoja et al. [47], could be a valuable extension to our method. It is crucial to acknowledge the sensitivity of Lagrange relaxation methods to the initialization of Lagrange multipliers and learning rates. Because of this, there is no assurance that the imitation policies can consistently meet the learned constraints. In our current implementation, we considered only hard constraints ($\alpha = 0$), assuming expert demonstrations are devoid of constraint violations. Recognizing that real-world expert agents may not always pursue constraint satisfaction, future research should explore extensions to model soft constraints [17, 25], where expert adherence to constraints is probabilistic. During initial training phases, our agent optimizes reward and entropy without accounting for expert knowledge, leading to inevitable constraint violations. Future investigations should prioritize safe exploration strategies, particularly in safety-critical environments. Our method, does also not guarantee safety when the agent deviates significantly from the optimal path. In such scenarios, uncertainty-aware methods [33, 34] are necessary to ensure safety. We assumed that the expert demonstrations are near optimal. This means that our method might, by mistake, assign higher costs to regions of the state space due to suboptimal demonstrations. To better accommodate these deviations from the expert demonstrations, increasing the budget parameter α could be a promising approach, although, this should still be investigated. Existing benchmarks predominantly focus on simple constraints, often characterized as location constraints. Future

work should emphasize the development of benchmarks featuring more intricate constraints, such as those based on the state of multiple objects or constraints defined by specific event sequences. The proposed method can be classified as an online ICRL algorithm as the constrained policy is learned through interaction with the environment. However, realistic applications often only have demonstration data available without a model of the environment. In light of recent advancements in offline Inverse Reinforcement Learning [48, 49, 50], extending ICRL to an offline training setting emerges as a critical and promising future direction.

References

[1] B. Christian. *The alignment problem: Machine learning and human values*. WW Norton & Company, UK, 2020.

[2] A. Balakrishnan, D. Bouneffouf, N. Mattei, and F. Rossi. *Incorporating behavioral constraints in online AI systems*. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, pages 3–11, 2019.

[3] S. Russell. *Human compatible: Artificial intelligence and the problem of control*. Penguin, UK, 2019.

[4] K. Van Moffaert and A. Nowé. *Multi-objective reinforcement learning using sets of pareto dominating policies*. The Journal of Machine Learning Research, 15(1):3483–3512, 2014.

[5] H. Mania, A. Guy, and B. Recht. *Simple random search provides a competitive approach to reinforcement learning*. arXiv preprint arXiv:1803.07055, 2018.

[6] S. Bhatnagar and K. Lakshmanan. *An online actor–critic algorithm with function approximation for constrained markov decision processes*. Journal of Optimization Theory and Applications, 153(3):688–708, 2012.

[7] C. Tessler, D. J. Mankowitz, and S. Mannor. *Reward constrained policy optimization*. arXiv preprint arXiv:1805.11074, 2018.

[8] J. Ho and S. Ermon. *Generative adversarial imitation learning*. Advances in neural information processing systems, 29, 2016.

[9] C. Finn, S. Levine, and P. Abbeel. *Guided cost learning: Deep inverse optimal control via policy optimization*. In International conference on machine learning, pages 49–58. PMLR, 2016.

[10] J. Fu, K. Luo, and S. Levine. *Learning robust rewards with adversarial inverse reinforcement learning*. arXiv preprint arXiv:1710.11248, 2017.

[11] D. R. Scobee and S. S. Sastry. *Maximum likelihood constraint inference for inverse reinforcement learning*. arXiv preprint arXiv:1909.05477, 2019.

[12] K. C. Stocking, D. L. McPherson, R. P. Matthew, and C. J. Tomlin. *Discretizing Dynamics for Maximum Likelihood Constraint Inference*, 2021. Available from: https://arxiv.org/abs/2109.04874.

[13] D. L. McPherson, K. C. Stocking, and S. S. Sastry. *Maximum Likelihood Constraint Inference from Stochastic Demonstrations*. In 2021 IEEE Conference on Control Technology and Applications (CCTA), pages 1208–1213. IEEE, 2021.

[14] M. Baert, S. Leroux, and P. Simoens. *Inverse reinforcement learning through logic constraint inference*. Machine Learning, pages 1–26, 2023.

[15] D. Papadimitriou, U. Anwar, and D. S. Brown. *Bayesian inverse constrained reinforcement learning*. In Workshop on Safe and Robust Control of Uncertain Systems (NeurIPS), 2021.

[16] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, et al. *Maximum entropy inverse reinforcement learning*. In Aaai, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[17] A. Glazier, A. Loreggia, N. Mattei, T. Rahgooy, F. Rossi, and B. Venable. *Learning Behavioral Soft Constraints from Demonstrations*, 2022. Available from: https://arxiv.org/abs/2202.10407.

[18] S. Malik, U. Anwar, A. Aghasi, and A. Ahmed. *Inverse constrained reinforcement learning*. In International Conference on Machine Learning, pages 7390–7399. PMLR, 2021.

[19] G. Liu, Y. Luo, A. Gaurav, K. Rezaee, and P. Poupart. *Benchmarking Constraint Inference in Inverse Reinforcement Learning*. arXiv preprint arXiv:2206.09670, 2022.

[20] B. D. Ziebart, J. A. Bagnell, and A. K. Dey. *Modeling interaction via the principle of maximum causal entropy*. In ICML, 2010.

[21] G. Qiao, G. Liu, P. Poupart, et al. *Multi-Modal Inverse Constrained Reinforcement Learning from a Mixture of Demonstrations*. In Thirty-seventh Conference on Neural Information Processing Systems, 2023.

[22] K. Kim, G. Swamy, Z. Liu, D. Zhao, S. Choudhury, and Z. S. Wu. *Learning Shared Safety Constraints from Multi-task Demonstrations*. 2023.

[23] M. Baert, S. Leroux, and P. Simoens. *Learning logic constraints from demonstration*. In NeSy2023: 17th International Workshop on Neural-Symbolic Learning and Reasoning, pages 78–84, 2023.

[24] D. Lindner, X. Chen, S. Tschiatschek, K. Hofmann, and A. Krause. *Learning Safety Constraints from Demonstrations with Unknown Rewards*. arXiv preprint arXiv:2305.16147, 2023.

[25] A. Gaurav, K. Rezaee, G. Liu, and P. Poupart. *Learning Soft Constraints From Constrained Expert Demonstrations*. arXiv preprint arXiv:2206.01311, 2022.

[26] M. Bain and C. Sammut. *A Framework for Behavioural Cloning*. In Machine Intelligence 15, pages 103–129, 1995.

[27] S. Ross, G. Gordon, and D. Bagnell. *A reduction of imitation learning and structured prediction to no-regret online learning*. In Proceedings of the fourteenth international conference on artificial intelligence and statistics, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[28] W. Chu and Z. Ghahramani. *Preference learning with Gaussian processes*. In Proceedings of the 22nd international conference on Machine learning, pages 137–144, 2005.

[29] K. Lee, L. Smith, A. Dragan, and P. Abbeel. *B-pref: Benchmarking preference-based reinforcement learning*. arXiv preprint arXiv:2111.03026, 2021.

[30] G. Chou, D. Berenson, and N. Ozay. *Learning constraints from demonstrations*. In Algorithmic Foundations of Robotics XIII: Proceedings of the 13th Workshop on the Algorithmic Foundations of Robotics 13, pages 228–245. Springer, 2020.

[31] G. Chou, N. Ozay, and D. Berenson. *Learning parametric constraints in high dimensions from demonstrations*. In Conference on Robot Learning, pages 1211–1230. PMLR, 2020.

[32] G. Chou, N. Ozay, and D. Berenson. *Learning constraints from locally-optimal demonstrations under cost function uncertainty*. IEEE Robotics and Automation Letters, 5(2):3682–3690, 2020.

[33] G. Chou, D. Berenson, and N. Ozay. *Uncertainty-aware constraint learning for adaptive safe motion planning from demonstrations*. In Conference on Robot Learning, pages 1612–1639. PMLR, 2021.

[34] G. Chou, H. Wang, and D. Berenson. *Gaussian process constraint learning for scalable chance-constrained motion planning from demonstrations*. IEEE Robotics and Automation Letters, 7(2):3827–3834, 2022.

[35] P. Abbeel and A. Y. Ng. *Apprenticeship learning via inverse reinforcement learning*. In Proceedings of the twenty-first international conference on Machine learning, page 1, 2004.

[36] E. Altman. *Constrained Markov decision processes: stochastic modeling*. Routledge, 1999.

[37] M. Wulfmeier, P. Ondruska, and I. Posner. *Maximum entropy deep inverse reinforcement learning*. arXiv preprint arXiv:1507.04888, 2015.

[38] A. Gleave and S. Toyer. *A Primer on Maximum Causal Entropy Inverse Reinforcement Learning*. arXiv preprint arXiv:2203.11409, 2022.

[39] S. Boyd, S. P. Boyd, and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[40] V. S. Borkar. *Stochastic approximation: a dynamical systems viewpoint*, volume 48. Springer, 2009.

[41] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. *Reinforcement learning with deep energy-based policies*. In International conference on machine learning, pages 1352–1361. PMLR, 2017.

[42] R. J. Williams. *Simple statistical gradient-following algorithms for connectionist reinforcement learning*. Machine learning, 8(3):229–256, 1992.

[43] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. *Proximal policy optimization algorithms*. arXiv preprint arXiv:1707.06347, 2017.

[44] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. *Openai gym*. arXiv preprint arXiv:1606.01540, 2016.

[45] R. Krajewski, J. Bock, L. Kloeker, and L. Eckstein. *The highD Dataset: A Drone Dataset of Naturalistic Vehicle Trajectories on German Highways for Validation of Highly Automated Driving Systems*. In 2018 21st International Conference on Intelligent Transportation Systems (ITSC), pages 2118–2125, 2018. doi:10.1109/ITSC.2018.8569552.

[46] X. Wang, H. Krasowski, and M. Althoff. *CommonRoad-RL: A Configurable Reinforcement Learning Environment for Motion Planning of Autonomous Vehicles*. In IEEE International Conference on Intelligent Transportation Systems (ITSC), 2021. doi:10.1109/ITSC48978.2021.9564898.

[47] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*. In International conference on machine learning, pages 1861–1870. PMLR, 2018.

[48] I. Kostrikov, O. Nachum, and J. Tompson. *Imitation Learning via Off-Policy Distribution Matching*. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net, 2020. Available from: https://openreview.net/forum?id=Hyg-JC4FDr.

[49] D. Garg, S. Chakraborty, C. Cundy, J. Song, and S. Ermon. *Iq-learn: Inverse soft-q learning for imitation*. Advances in Neural Information Processing Systems, 34:4028–4039, 2021.

[50] A. J. Chan and M. van der Schaar. *Scalable Bayesian Inverse Reinforcement Learning*. In International Conference on Learning Representations, 2021. Available from: https://openreview.net/forum?id=4qR3coiNaIv.

[51] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. *High-dimensional continuous control using generalized advantage estimation*. arXiv preprint arXiv:1506.02438, 2015.

[52] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, Cambridge, 2018.

[53] D. P. Kingma and J. Ba. *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980, 2014.

2.7 Appendix

2.7.1 Causal Entropy

In this section we elaborate on why it is not advisable to adopt the principle of maximum non-causal entropy (i.e., Shannon entropy) in domains which are characterized by stochastic transition dynamics. Consider the following derivation from the definition of the Shannon entropy of a trajectory:

$$\sum_{t=0}^{T-1} \gamma^t H(s_t, a_t \mid s_{t-1}, a_{t-1}) \tag{2.14}$$

$$= \sum_{t=0}^{T-1} \gamma^t H(s_t \mid s_{t-1}, a_{t-1}) + \sum_{t=0}^{T-1} \gamma^t H(a_t \mid s_{t-1}, a_{t-1}) \tag{2.15}$$

$$= \sum_{t=0}^{T-1} \gamma^t H(s_t \mid s_{t-1}, a_{t-1}) + \sum_{t=0}^{T-1} \gamma^t H(a_t \mid s_{t-1}). \tag{2.16}$$

The non-causal entropy can be written as the sum of the entropy of the transition dynamics (first term of eq. (2.16)) and the causal entropy (second term of eq. (2.16)). Due to the dependency on transition dynamics, the maximization of Shannon entropy introduces a bias towards actions with uncertain (and potentially hazardous) outcomes. Therefore, in place of conventional Shannon entropy, it is recommended to utilize causal entropy in decision-making processes to mitigate such biases.

2.7.2 Minimizing Policy Gradient Variance

The vanilla policy gradient is characterized by minimal bias but tends to exhibit high variance. To mitigate this variance while maintaining low bias, the introduction of a state-dependent baseline is a well-established technique [42].

Proposition 6 *In the gradient of the Lagrangian (eq.* (2.11)*), we can replace* $\sum_{t'=t}^{T-1} \gamma^{t'-t}\beta$ *with a state dependent baseline* $b(s_t)$*. Proof. See Section 2.7.3.5 in the appendix.*

Because of proposition 6 we can replace $\sum_{t'=t}^{T-1} \gamma^{t'-t}\beta$ with the state-value function $V^{\pi_\theta}(s_t)$ which is a commonly used baseline. The gradient can then

be written as

$$\nabla_\theta \mathcal{L}(\theta, \lambda) = \mathbb{E}_{\tau \sim \pi_\theta} \Bigg[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \big(Q^{\pi_\theta}(s_t, a_t) - \beta \log \pi_\theta(a_t \mid s_t) - V^{\pi_\theta}(s_t) \big) \Bigg] \tag{2.17}$$

$$= \mathbb{E}_{\tau \sim \pi_\theta} \Bigg[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) A^{\pi_\theta}(s_t, a_t) \Bigg] \tag{2.18}$$

with $A^{\pi_\theta}(s, a)$ the advantage function. We adopt generalized advantage estimation (GAE) [51] to obtain an approximation of the advantage.

2.7.3 Proofs

2.7.3.1 Proposition 2

The primal objective is defined as

$$\max_{\pi \in \Pi} \mathcal{L}(\pi, \lambda) \tag{2.19}$$

with $\mathcal{L}(\pi, \lambda)$ the Lagrangian defined in eq. (2.4), and thus the optimal policy

$$\pi^* = \arg\max_{\pi \in \Pi} \mathcal{L}(\pi, \lambda). \tag{2.20}$$

The simplex Π which restricts the valid policies is defined by two constraints (see eq. (2.3)). The first constraint requires that $\pi(a \mid s)$ is positive for all states and timesteps. We treat this constraint as implicit since the causal entropy is not defined for negative probabilities. The second constraint requires that π is normalized over all actions for all states and timesteps. We assume, for now, the state-action space is discrete. Augmenting the normalization constraint on the current objective gives us:

$$\begin{aligned} \pi^* &= \arg\max_{\pi \in \Pi} \mathcal{L}(\pi, \lambda) \\ \text{s.t.} \sum_{t=0}^{T-1} \sum_{s_t \in \mathcal{S}} &\left(\sum_{a \in \mathcal{A}} \pi(a \mid s_t) \; - 1 \right) = 0. \end{aligned} \tag{2.21}$$

Then, the Lagrangian of the optimization problem in eq. (2.21) is defined as

$$\Psi(\pi, \lambda, \mu) = \mathcal{L}(\pi, \lambda) + \sum_{t=0}^{T-1} \sum_{s_t \in \mathcal{S}} \mu_{s_t} \left(\sum_{a \in \mathcal{A}} \pi(a \mid s) \; - 1 \right) \tag{2.22}$$

with $\{\mu_{s_t}\}_{s_t \in \mathcal{S}, 0 \leq t \leq T-1}$ the dual variables for the normalization constraint. Then the optimal policy is

$$\pi^* = \arg\max_{\pi} \Psi(\pi, \lambda, \mu). \tag{2.23}$$

The optimal policy π^* satisfies the KKT condition:

$$\nabla_\pi \Psi(\pi^*, \lambda, \mu) = 0. \tag{2.24}$$

To obtain an expression for the optimal policy, we take the gradient of the Lagrangian w.r.t. the policy, equate $\nabla_\pi \Psi(\pi, \lambda, \mu)$ to zero and solve for π.

Proposition 7 *The gradient of the Lagrangian (eq.* (2.22)*) w.r.t. to the policy $\pi(a_t \mid s_t)$ is given by*

$$\begin{aligned}\nabla_{\pi(a_t|s_t)} \Psi(\pi, \lambda, \mu) = \mu_{s_t} + \rho_\pi(s_t) \Bigg(& R(s_t, a_t) - \lambda \cdot \phi(s_t, a_t) \\ & - \beta \log \pi(a_t \mid s_t) - \beta + \mathbb{E}_\pi \Bigg[\sum_{t'=t+1}^{T-1} \gamma^{t'-t} \big(R(S_{t'}, A_{t'}) \\ & - \lambda \cdot \phi(S_{t'}, A_{t'}) - \beta \log \pi(A_{t'} \mid S_{t'}) \big) \Big| s_t, a_t \Bigg] \Bigg). \end{aligned} \tag{2.25}$$

Proof. We take the gradient of each term of the Lagrangian (eq. (2.22)*) w.r.t. to the policy $\pi(a_t \mid s_t)$ separately. Uppercase characters S_t, A_t represent random variables while lower case characters s_t, a_t denote individual observations.*

$$\nabla_{\pi(a_t|s_t)} \mathbb{E}_\pi \left[R(\tau) \right] \tag{2.26}$$

$$= \nabla_{\pi(a_t|s_t)} \mathbb{E}_\pi \left[\sum_{t'=0}^{T-1} \gamma^{t'} R(S_{t'}, A_{t'}) \right] \tag{2.27}$$

$$= \mathbb{E}_{S_t \sim \pi} \left[\nabla_{\pi(a_t|s_t)} \mathbb{E}_\pi \left[\sum_{t'=0}^{T-1} \gamma^{t'} R(S_{t'}, A_{t'}) \Big| S_t \right] \right] \tag{2.28}$$

$$= \mathbb{E}_{S_t \sim \pi} \left[\gamma^t \mathbb{I}[S_t = s_t] \nabla_{\pi(a_t|s_t)} \mathbb{E}_\pi \left[\sum_{t'=t}^{T-1} \gamma^{t'-t} R(S_{t'}, A_{t'}) \Big| S_t = s_t \right] \right] \tag{2.29}$$

$$= \rho_\pi(s_t) \nabla_{\pi(a_t|s_t)} \mathbb{E}_\pi \left[\sum_{t'=t}^{T-1} \gamma^{t'-t} R(S_{t'}, A_{t'}) \Big| S_t = s_t \right] \tag{2.30}$$

$$\begin{aligned} = \rho_\pi(s_t) \nabla_{\pi(a_t|s_t)} \Bigg(& \pi(a_t \mid s_t) \mathbb{E}_\pi \left[\sum_{t'=t}^{T-1} \gamma^{t'-t} R(S_{t'}, A_{t'}) \Big| S_t = s_t, A_t = a_t \right] \\ & + \Bigg(\sum_{a'_t \neq a_t} \pi(a'_t \mid s_t) \Bigg) \mathbb{E}_\pi \left[\sum_{t'=t}^{T-1} \gamma^{t'-t} R(S_{t'}, A_{t'}) \Big| S_t = s_t, A_t \neq a_t \right] \Bigg) \end{aligned} \tag{2.31}$$

$$= \rho_\pi(s_t) \mathbb{E}_\pi \left[\sum_{t'=t}^{T-1} \gamma^{t'-t} R(S_{t'}, A_{t'}) \Big| S_t = s_t, A_t = a_t \right] \tag{2.32}$$

$$= \rho_\pi(s_t) \left[R(s_t, a_t) + \mathbb{E}_\pi \left[\sum_{t'=t+1}^{T-1} \gamma^{t'-t} R(S_{t'}, A_{t'}) \Big| s_t, a_t \right] \right], \tag{2.33}$$

where $\rho_\pi(s_t)$ represents the discounted probability that an agent acting according to policy π will be in state s_t at time t

$$\rho_\pi(s_t) = \mathbb{E}_{S_t \sim \pi}\left[\gamma^t \mathbb{I}[S_t = s_t]\right]. \tag{2.34}$$

The gradient of the entropy term:

$$\nabla_{\pi(a_t|s_t)} \mathbb{E}_\pi\left[H(\tau)\right] \tag{2.35}$$

$$= \nabla_{\pi(a_t|s_t)} \mathbb{E}_\pi\left[\sum_{t'=0}^{T-1} \gamma^{t'} H(A_{t'} \mid S_{t'})\right] \tag{2.36}$$

$$= -\nabla_{\pi(a_t|s_t)} \mathbb{E}_\pi\left[\sum_{t'=0}^{T-1} \gamma^{t'} \log \pi(A_{t'} \mid S_{t'})\right] \tag{2.37}$$

$$= -\rho_\pi(s_t) \nabla_{\pi(a_t|s_t)} \mathbb{E}_\pi\left[\sum_{t'=t}^{T-1} \gamma^{t'-t} \log \pi(A_{t'} \mid S_{t'}) \middle| S_t = s_t\right] \tag{2.38}$$

$$= -\rho_\pi(s_t) \nabla_{\pi(a_t|s_t)} \mathbb{E}_\pi\Bigg[\log \pi(A_t \mid S_t) + \sum_{t'=t+1}^{T-1} \gamma^{t'-t} \log \pi(A_{t'} \mid S_{t'}) \Big| S_t = s_t\Bigg] \tag{2.39}$$

$$= -\rho_\pi(s_t)\Bigg(1 + \log \pi(a_t \mid s_t) + \mathbb{E}_\pi\left[\sum_{t'=t+1}^{T-1} \gamma^{t'-t} \log \pi(A_{t'} \mid S_{t'}) \middle| s_t, a_t\right]\Bigg). \tag{2.40}$$

The gradient of the feature matching term:

$$\nabla_{\pi(a_t|s_t)} \lambda \cdot \left(\mathbb{E}_\mathcal{D}\left[\sum_{t'=0}^{T-1} \gamma^{t'} \phi(S_t, A_t)\right] - \mathbb{E}_\pi\left[\sum_{t'=0}^{T-1} \gamma^{t'} \phi(S_{t'}, A_{t'})\right]\right) \tag{2.41}$$

$$= -\nabla_{\pi(a_t|s_t)} \lambda \cdot \mathbb{E}_\pi\left[\sum_{t'=0}^{T-1} \gamma^{t'} \phi(S_{t'}, A_{t'})\right] \tag{2.42}$$

$$= -\rho_\pi(s_t) \lambda \cdot \nabla_{\pi(a_t|s_t)} \mathbb{E}_\pi\left[\sum_{t'=0}^{T-1} \gamma^{t'-t} \phi(S_{t'}, A_{t'}) \middle| S_t = s_t\right] \tag{2.43}$$

$$= -\rho_\pi(s_t) \lambda \cdot \left[\phi(s_t, a_t) + \mathbb{E}_\pi\left[\sum_{t'=t+1}^{T-1} \gamma^{t'-t} \phi(S_{t'}, A_{t'}) \middle| s_t, a_t\right]\right]. \tag{2.44}$$

The gradient of the normalization constraint term:

$$\nabla_{\pi(a_t|s_t)} \sum_{t=0}^{T-1} \sum_{s_t \in \mathcal{S}} \mu_{s_t} \left(\sum_{a \in \mathcal{A}} \pi(a \mid s) - 1\right) \tag{2.45}$$

$$= \mu_{s_t}. \tag{2.46}$$

After equating the gradient from Proposition 7 to zero and re-arranging, we get an expression for the optimal policy π^*:

$$\pi^*(a_t \mid s_t) = \exp\left(\frac{1}{\beta}\Bigg(R(s_t, a_t) - \lambda \cdot \phi(s_t, a_t) + \frac{\mu_s}{\rho_\pi(s_t)} - \beta + \mathbb{E}_\pi\Bigg[\sum_{t'=t+1}^{T-1} \gamma^{t'-t}\big(R(S_{t'}, A_{t'}) - \lambda \cdot \phi(S_{t'}, A_{t'}) - \beta \log \pi(A_{t'} \mid S_{t'})\big)\Bigg| s_t, a_t\Bigg]\Bigg)\right). \quad (2.47)$$

We define $V(s_t)$ and $Q(s_t, a_t)$ as

$$V(s_t) \triangleq \frac{-\mu_{s_t}}{\rho_\pi(s_t)} + \beta \quad (2.48)$$

$$Q(s_t, a_t) \triangleq R(s_t, a_t) - \lambda \cdot \phi(s_t, a_t) + \mathbb{E}_\pi\Bigg[\sum_{t'=t+1}^{T-1} \gamma^{t'-t}\big(R(S_{t'}, A_{t'}) - \lambda \cdot \phi(S_{t'}, A_{t'}) - \beta \log \pi(A_{t'} \mid S_{t'})\Bigg| s_t, a_t\Bigg]. \quad (2.49)$$

The optimal policy can then be described as

$$\pi^*(a_t \mid s_t) = \exp\left(\frac{1}{\beta}\big(Q(s_t, a_t) - V(s_t)\big)\right). \quad (2.50)$$

In eq. (2.3) we defined the set of valid policies as the set of normalized policies Π and assume all values of μ are chosen such that $\sum_{a \in \mathcal{A}} \pi(a \mid s) = 1$ $\forall s \in \mathcal{S}$. Because of this, we can say that $\forall s \in \mathcal{S}$:

$$1 = \sum_{a \in \mathcal{A}} \pi(a \mid s) \quad (2.51)$$

$$= \sum_{a \in \mathcal{A}} \exp\left(\frac{1}{\beta}(Q(s, a) - V(s))\right) \quad (2.52)$$

$$\exp(\frac{1}{\beta}V(s)) = \sum_{a \in \mathcal{A}} \exp\left(\frac{1}{\beta}(Q(s, a) - V(s))\right) \exp(\frac{1}{\beta}V(s)) \quad (2.53)$$

$$= \sum_{a \in \mathcal{A}} \exp\left(\frac{1}{\beta}Q(s, a)\right) \quad (2.54)$$

$$V(s) = \beta \log \sum_{a \in \mathcal{A}} \exp\left(\frac{1}{\beta}Q(s, a)\right). \quad (2.55)$$

For a continuous action space we get

$$V(s) = \beta \log \int \exp\left(\frac{1}{\beta}Q(s, a)\right) \mathrm{d}a. \quad (2.56)$$

Similarly, it can be demonstrated that

$$
\begin{aligned}
&Q(s_t, a_t) \\
&= R(s_t, a_t) - \lambda \cdot \phi(s_t, a_t) + \mathbb{E}_\pi \Bigg[\sum_{t'=t+1}^{T-1} \gamma^{t'-t} \big(R(S_{t'}, A_{t'}) \\
&\qquad\qquad - \lambda \cdot \phi(S_{t'}, A_{t'}) - \beta \log \pi(A_{t'} \mid S_{t'}) \big) \Big| s_t, a_t \Bigg] \qquad (2.57) \\
&= R(s_t, a_t) - \lambda \cdot \phi(s_t, a_t) \\
&\qquad + \gamma \, \mathbb{E}_p \Big[\mathbb{E}_\pi \Big[Q(S_{t+1}, A_{t+1}) - \beta \log \pi(A_{t+1} \mid S_{t+1}) \Big] \Big| s_t, a_t \Big] \qquad (2.58) \\
&= R(s_t, a_t) - \lambda \cdot \phi(s_t, a_t) + \gamma \, \mathbb{E}_p \Big[\mathbb{E}_\pi \Big[Q(S_{t+1}, A_{t+1}) \\
&\qquad\qquad - \big(Q(S_{t+1}, A_{t+1}) - V(S_{t+1}) \big) \Big] \Big| s_t, a_t \Big] \qquad (2.59) \\
&= R(s_t, a_t) - \lambda \cdot \phi(s_t, a_t) + \gamma \, \mathbb{E}_p \Big[V(S_{t+1}) \Big| s_t, a_t \Big]. \qquad (2.60)
\end{aligned}
$$

2.7.3.2 Theorem 3

First, we will prove the convergence of the policy evaluation step.

Proposition 8 (Policy Evaluation) *Starting with an initial Q-function $Q^0 : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ with $|\mathcal{A}| < \infty$, where $Q^{k+1} = \mathcal{T}^\pi Q^k$. Then the sequence Q^k will converge to the Q-value of π as $k \to \infty$.*

Proof.
We define an augmented reward signal as

$$R_\pi(s_t, a_t) \triangleq R(s_t, a_t) - \lambda \cdot \phi(s_t, a_t) + \beta H(a_t \mid s_t). \qquad (2.61)$$

Applying the standard convergence results for policy evaluation [52] proves the proposition. It is necessary to have $|\mathcal{A}| < \infty$ in order to ensure that the augmented reward is bounded.

For the projection presented in eq. (2.10), we can prove that the new, projected policy has a higher value than the old policy according to the objective in eq. (2.4).

Proposition 9 (Policy Improvement) *Given $\pi_{old} \in \Pi$ and π_{new} be the policy obtained by solving the minimization problem defined in eq.* (2.10)*. Then $Q^{\pi_{new}}(s_t, a_t) \geq Q^{\pi_{old}}(s_t, a_t)$ for all $(s_t, a_t) \in \mathcal{S} \times \mathcal{A}$ with $|\mathcal{A}| < \infty$.*

Proof.
Let $Q^{\pi_{old}}$ and $V^{\pi_{old}}$ be the action-value and state-value function corresponding to $\pi_{old} \in \Pi$. We can define $J_{\pi_{old}}$ as

$$J_{\pi_{old}}(\pi'(\cdot \mid s_t)) = D_{KL}\left(\pi' \mid\mid \exp(\frac{1}{\beta}\left(Q^{\pi_{old}}(s_t \mid \cdot) - V^{\pi_{old}}(s_t)\right))\right). \quad (2.62)$$

Then π_{new} can be defined as

$$\pi_{new}(\cdot \mid s_t) = \arg\min_{\pi' \in \Pi} J_{\pi_{old}}(\pi'(\cdot \mid s_t)). \quad (2.63)$$

The inequality $J_{\pi_{old}}(\pi_{new}(\cdot \mid s_t)) \leq J_{\pi_{old}}(\pi_{old}(\cdot \mid s_t))$ must hold, as we have the option of always selecting $\pi_{new} = \pi_{old}$ from the set of policies Π. Hence,

$$\begin{aligned}&\mathbb{E}_{\pi_{new}}\left[\log \pi_{new}(a_t \mid s_t) - \frac{1}{\beta}(Q^{\pi_{old}}(s_t, a_t) - V^{\pi_{old}}(s_t))\right] \leq \\ &\mathbb{E}_{\pi_{old}}\left[\log \pi_{old}(a_t \mid s_t) - \frac{1}{\beta}(Q^{\pi_{old}}(s_t, a_t) - V^{\pi_{old}}(s_t))\right] \quad (2.64)\end{aligned}$$

$$\begin{aligned}&\mathbb{E}_{\pi_{new}}\left[\log \pi_{new}(a_t \mid s_t) - \frac{1}{\beta}Q^{\pi_{old}}(s_t, a_t)\right] + \frac{1}{\beta}V^{\pi_{old}}(s_t) \leq \\ &\mathbb{E}_{\pi_{old}}\left[\log \pi_{old}(a_t \mid s_t) - \frac{1}{\beta}Q^{\pi_{old}}(s_t, a_t)\right] + \frac{1}{\beta}V^{\pi_{old}}(s_t) \quad (2.65)\end{aligned}$$

$$\begin{aligned}&\mathbb{E}_{\pi_{new}}\left[\log \pi_{new}(a_t \mid s_t) - \frac{1}{\beta}Q^{\pi_{old}}(s_t, a_t)\right] \leq \\ &\mathbb{E}_{\pi_{old}}\left[\log \pi_{old}(a_t \mid s_t) - \frac{1}{\beta}Q^{\pi_{old}}(s_t, a_t)\right] \quad (2.66)\end{aligned}$$

$$\mathbb{E}_{\pi_{new}}\left[Q^{\pi_{old}}(s_t, a_t) - \beta \log \pi_{new}(a_t \mid s_t)\right] \geq V^{\pi_{old}}(s_t). \quad (2.67)$$

We can bring $V^{\pi_{old}}$ outside the expectation since the state-value function only depends on the current state (eq. (2.65)*). Eq.* (2.67) *follows from the definition of the optimal policy in eq.* (2.6)*. We can repeatedly expand the Bellman equation by applying the Bellman equation and the inequality of eq.* (2.67)*:*

$$\begin{aligned}&Q^{\pi_{old}}(s_t, a_t) \\ &= R(s_t, a_t) - \lambda \cdot \phi(s_t, a_t) + \gamma\, \mathbb{E}_p\left[V^{\pi_{old}}(s_{t+1})\right] \quad (2.68)\\ &\leq R(s_t, a_t) - \lambda \cdot \phi(s_t, a_t) + \gamma\, \mathbb{E}_{p,\pi_{new}}\Big[\\ &\qquad Q^{\pi_{old}}(s_{t+1}, a_{t+1}) - \beta \log \pi_{new}(a_{t+1} \mid s_{t+1})\Big] \quad (2.69)\\ &\vdots \\ &\leq Q^{\pi_{new}}(s_t, a_t). \quad (2.70)\end{aligned}$$

Then, the convergence to $Q^{\pi_{new}}$ follows from Proposition 8.

At each iteration i, the policy π_i will reach an optimum due to Proposition 9. Because the sequence Q^{π_i} increases monotonically and is bounded for $\pi \in \Pi$, the sequence converges to some π^*. We have to prove that π^* is indeed optimal. At convergence, it must be the case that $J_{\pi^*}(\pi^*(\cdot \mid s_t)) < J_{\pi^*}(\pi(\cdot \mid s_t))$ for all $\pi \in \Pi$, $\pi \neq \pi^*$. By repeatedly expanding Q^{π^*} as done in Proposition 9, we can show that $Q^{\pi^*}(s_t, a_t) > Q^{\pi}(s_t, a_t)$ for all $(s_t, a_t) \in \mathcal{S} \times \mathcal{A}$. Thus, the value of any other policy in Π is lower than that of the converged policy. This makes π^* optimal in Π.

2.7.3.3 Proposition 5

$\max_{\pi \in \Pi} \mathcal{L}(\pi, \lambda)$ can be viewed as a pointwise maximum of affine functions of λ, thus is concave. $\lambda \geq 0$ is an affine constraint. Hence, the dual problem (eq. (2.5)) is a convex optimization problem.

2.7.3.4 Proposition 4

We calculate the gradient of each term of the Lagrangian eq. (2.4) separately w.r.t. the policy parameters θ. The gradient of the causal entropy term is

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}\big[H(\tau)\big] \tag{2.71}$$

$$= \nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1}\gamma^t H(a_t \mid s_t)\right] \tag{2.72}$$

$$= -\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1}\gamma^t \log \pi_\theta(a_t \mid s_t)\right] \tag{2.73}$$

$$= -\nabla_\theta \sum_\tau \left(P(\tau \mid \pi_\theta)\sum_{t=0}^{T-1}\gamma^t \log \pi_\theta(a_t \mid s_t)\right) \tag{2.74}$$

$$= -\sum_\tau \Bigg(\nabla_\theta P(\tau \mid \pi_\theta)\sum_{t=0}^{T-1}\gamma^t \log \pi_\theta(a_t \mid s_t) + P(\tau \mid \pi_\theta)\sum_{t=0}^{T-1}\gamma^t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\Bigg) \tag{2.75}$$

$$= -\sum_\tau \Bigg(P(\tau \mid \pi_\theta)\nabla_\theta \log P(\tau \mid \pi_\theta)\sum_{t=0}^{T-1}\gamma^t \log \pi_\theta(a_t \mid s_t) + P(\tau \mid \pi_\theta)\sum_{t=0}^{T-1}\gamma^t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\Bigg) \tag{2.76}$$

$$= -\sum_\tau P(\tau \mid \pi_\theta)\Bigg(\nabla_\theta \log P(\tau \mid \pi_\theta)\sum_{t=0}^{T-1}\gamma^t \log \pi_\theta(a_t \mid s_t) + \sum_{t=0}^{T-1}\gamma^t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\Bigg) \tag{2.77}$$

$$= -\mathbb{E}_{\tau\sim\pi_\theta}\Bigg[\nabla_\theta \log P(\tau \mid \pi_\theta)\sum_{t=0}^{T-1}\gamma^t \log \pi_\theta(a_t \mid s_t) + \sum_{t=0}^{T-1}\gamma^t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\Bigg] \tag{2.78}$$

$$= -\mathbb{E}_{\tau\sim\pi_\theta}\Bigg[\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t \mid s_t)\sum_{t'=t}^{T-1}\gamma^{t'-t} \log \pi_\theta(a_{t'} \mid s_{t'}) + \sum_{t=0}^{T-1}\gamma^t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\Bigg] \tag{2.79}$$

$$= -\mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t \mid s_t)\left(\sum_{t'=t}^{T-1}\gamma^{t'-t} \log \pi_\theta(a_{t'} \mid s_{t'}) + 1\right)\right]. \tag{2.80}$$

In eq. (2.72) and eq. (2.73), we apply the definition of causal entropy. $P(\tau \mid \pi)$ denotes the probability of a trajectory τ under policy π. Eq. (2.76) follows from $\nabla P(\tau \mid \pi) = P(\tau \mid \pi)\nabla \log P(\tau \mid \pi)$. Eq. (2.79) follows from the definition of the probability of a trajectory τ under a policy π: $P(\tau \mid \pi) = \mathcal{I}(s_0)\prod_{t=0}^{T-1} p(s_{t+1} \mid s_t, a_t)\pi(a_t \mid s_t)$. Then the second sum needs to be updated since the policy at time t' cannot affect the policy at time t when $t < t'$ because of the causal relation between consecutive states. The gradient of the reward term can be calculated similarly

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi_\theta}\big[R(\tau)\big] \tag{2.81}$$

$$= \mathbb{E}_{\tau\sim\pi_\theta}\Bigg[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t) \sum_{t'=t}^{T-1} \gamma^{t'-t} R(s_t, a_t)\Bigg], \tag{2.82}$$

and the feature matching term:

$$\nabla_\theta \lambda \cdot \Bigg(\mathbb{E}_{\tau\sim\mathcal{D}}\Bigg[\sum_{t=0}^{T-1} \gamma^t \phi(s_t, a_t)\Bigg] - \mathbb{E}_{\tau\sim\pi_\theta}\Bigg[\sum_{t=0}^{T-1} \gamma^t \phi(s_t, a_t)\Bigg] - \boldsymbol{\alpha}_k\Bigg) \tag{2.83}$$

$$= -\nabla_\theta \lambda \cdot \mathbb{E}_{\tau\sim\pi_\theta}\Bigg[\sum_{t=0}^{T-1} \gamma^t \phi(s_t, a_t)\Bigg] \tag{2.84}$$

$$= -\mathbb{E}_{\tau\sim\pi_\theta}\Bigg[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t) \sum_{t'=t}^{T-1} \gamma^{t'-t} \lambda \cdot \phi(s_t, a_t)\Bigg]. \tag{2.85}$$

In our work, the same discount factor is used to discount rewards, costs and entropies. Putting it all together, we get

$$\nabla_\theta \mathcal{L}(\theta, \lambda) = \mathbb{E}_{\tau\sim\pi_\theta}\Bigg[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \sum_{t'=t}^{T-1} \gamma^{t'-t} \Big(R(s_{t'}, a_{t'}) - \lambda \cdot \phi(s_{t'}, a_{t'}) - \beta \log \pi_\theta(a_{t'} \mid s_{t'}) - \beta\Big)\Bigg]. \tag{2.86}$$

We rewrite eq. (2.86)

$$\begin{aligned}\nabla_\theta \mathcal{L}(\theta, \lambda) &= \mathbb{E}_{\tau\sim\pi_\theta}\Bigg[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\Bigg(R(s_t, a_t) - \lambda \cdot \phi(s_t, a_t) \\ &+ \sum_{t'=t+1}^{T-1} \gamma^{t'-t}\big(R(s_{t'}, a_{t'}) - \lambda \cdot \phi(s_{t'}, a_{t'}) - \beta \log \pi_\theta(a_{t'} \mid s_{t'})\big) \\ &\qquad - \beta \log \pi_\theta(a_t \mid s_t) - \sum_{t'=t}^{T-1} \gamma^{t'-t} \beta\Bigg)\Bigg].\end{aligned} \tag{2.87}$$

Using the definition of the action-value function (eq. (2.7)) we obtain:

$$\nabla_\theta \mathcal{L}(\theta, \lambda) = \mathbb{E}_{\tau \sim \pi_\theta}\Bigg[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \big(Q(s_t, a_t) - \beta \log \pi_\theta(a_t \mid s_t) - \sum_{t'=t}^{T-1} \gamma^{t'-t} \beta\big)\Bigg]. \quad (2.88)$$

2.7.3.5 Proposition 6

We can replace $\sum_{t'=t}^{T-1} \gamma^{t'-t}\beta$ with a state dependent baseline $b(s_t)$ in eq. (2.11) only when this does not affect the gradient $\nabla_\theta \mathcal{L}(\theta, \lambda)$. We rewrite eq. (2.11):

$$\nabla_\theta \mathcal{L}(\theta, \lambda) \quad (2.89)$$

$$= \mathbb{E}_{\tau \sim \pi_\theta}\Bigg[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \big(Q(s_t, a_t) - \beta \log \pi_\theta(a_t \mid s_t) - \sum_{t'=t}^{T-1} \gamma^{t'-t} \beta\big)\Bigg] \quad (2.90)$$

$$= \mathbb{E}_{\tau \sim \pi_\theta}\Bigg[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \big(Q(s_t, a_t) - \beta \log \pi_\theta(a_t \mid s_t)\big)\Bigg] - \mathbb{E}_{\tau \sim \pi_\theta}\Bigg[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t) \sum_{t'=t}^{T-1} \gamma^{t'-t} \beta\Bigg] \quad (2.91)$$

In the second term, we replace $\sum_{t'=t}^{T-1} \gamma^{t'-t}\beta$ with a state-dependent baseline $b(s_t)$ which gives us:

$$\nabla_\theta \mathcal{L}(\theta, \lambda) = \mathbb{E}_{\tau \sim \pi_\theta}\Bigg[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \big(Q(s_t, a_t) - \beta \log \pi_\theta(a_t \mid s_t)\big)\Bigg] - \mathbb{E}_{\tau \sim \pi_\theta}\Bigg[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t) b(s_t)\Bigg] \quad (2.92)$$

Since the introduced baseline only affects the second term it suffices to prove that

$$\mathbb{E}_{\tau \sim \pi_\theta}\Bigg[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t) \sum_{t'=t}^{T-1} \gamma^{t'-t} \beta\Bigg] = \mathbb{E}_{\tau \sim \pi_\theta}\Bigg[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t) b(s_t)\Bigg] \quad (2.93)$$

$$\mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t \mid s_t) b(s_t)\right] \tag{2.94}$$

$$= \mathbb{E}_{s_{0:T-1},a_{0:T-1}}\left[\sum_{t=0}^{T-1}\nabla_\theta \log \pi_\theta(a_t \mid s_t) b(s_t)\right] \tag{2.95}$$

$$= \mathbb{E}_{s_{0:t},a_{0:t-1}}\left[\sum_{t=0}^{T-1}\mathbb{E}_{s_{t+1:T-1},a_{t:T-1}}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) b(s_t)\right]\right] \tag{2.96}$$

$$= \mathbb{E}_{s_{0:t},a_{0:t-1}}\left[\sum_{t=0}^{T-1} b(s_t)\,\mathbb{E}_{s_{t+1:T-1},a_{t:T-1}}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right]\right] \tag{2.97}$$

$$= \mathbb{E}_{s_{0:t},a_{0:t-1}}\left[\sum_{t=0}^{T-1} b(s_t)\,\mathbb{E}_{a_t}\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right]\right] \tag{2.98}$$

$$= \mathbb{E}_{s_{0:t},a_{0:t-1}}\left[\sum_{t=0}^{T-1} b(s_t)\sum_{a_t\in\mathcal{A}}\pi_\theta(a_t \mid s_t)\nabla_\theta \log \pi_\theta(a_t \mid s_t)\right]\right] \tag{2.99}$$

$$= \mathbb{E}_{s_{0:t},a_{0:t-1}}\left[\sum_{t=0}^{T-1} b(s_t)\sum_{a_t\in\mathcal{A}}\nabla_\theta \pi_\theta(a_t \mid s_t)\right]\right] \tag{2.100}$$

$$= \mathbb{E}_{s_{0:t},a_{0:t-1}}\left[\sum_{t=0}^{T-1} b(s_t)\nabla_\theta \sum_{a_t\in\mathcal{A}}\pi_\theta(a_t \mid s_t)\right]\right] \tag{2.101}$$

$$= \mathbb{E}_{s_{0:t},a_{0:t-1}}\left[\sum_{t=0}^{T-1} b(s_t)\nabla_\theta \cdot 1\right]\right] \tag{2.102}$$

$$= 0 \tag{2.103}$$

$\mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1}\nabla_\theta \log \pi(a_t \mid s_t)\sum_{t'=t}^{T-1}\gamma^{t'-t}\beta\right]$ can be solved similarly and also results in 0.

2.7.4 Details on the Environments

In this section we provide details on and screenshots from the environments used for evaluation.

2.7.4.1 Gridworld

Figure 2.5 shows the gridworld used in the experiments, with the yellow "I" representing the initial state, the yellow "O" the goal state and the red crosses the constrained states.

2.7.4.2 Virtual Robotics Environment

Figure 2.6 shows screenshots of the realistic robotic environments.

2.7.4.3 Realistic Traffic Environment

Figure 2.7 shows the realistic traffic environment (image from original publication [19]). The ego car is shown in blue, other cars are red. The

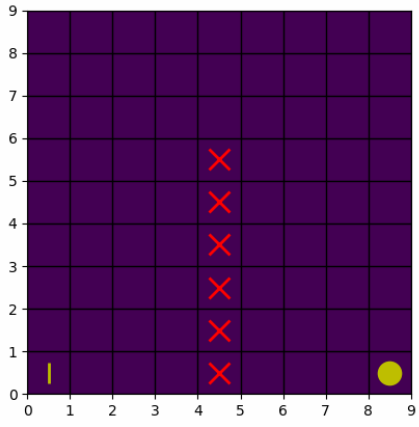
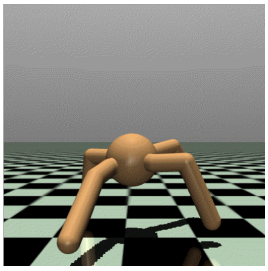
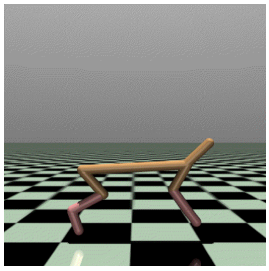


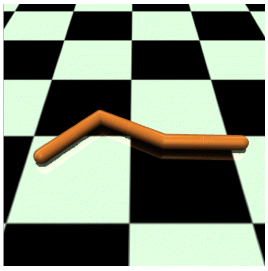
Figure 2.5: Gridworld environment



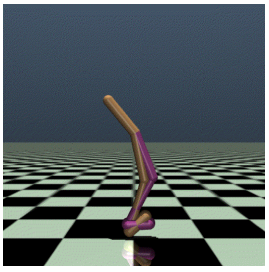
(a) Ant



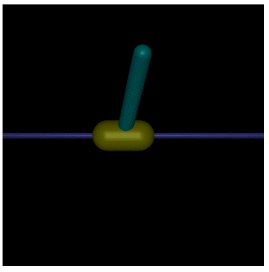
(b) Half-cheetah



(c) Swimmer



(d) Walker



(e) Inverted pendulum

Figure 2.6: Screenshots from the virtual robotics environments.

Figure 2.7: Realistic traffic environment

agent only has partial state information, observations are restricted by an observation radius (visualized in blue around the ego agent). The agent's destination is shown in yellow.

2.7.5 Experimental Settings

All experiments were run on a NVIDIA GeForce GTX 980 GPU with 3 GB RAM. We adopted Adam [53] to optimize all neural networks. To calculate the nominal policy, we use Proximal Policy Optimization (PPO) [43]. The output layer of the policy network utilizes a soft-max activation function such that the requirements of the policy specified in eq. 2.3 are met.

The ϕ_ζ network is a standard feedforward neural network with ReLU activation on the hidden layers and Sigmoid activation on the output layer. λ and ζ are adjusted using possibly different learning rates. Every iteration the learning rate for ζ is updated using an exponential decay schedule parameterized by a decay factor. Table 2.2 and 2.3 report an overview of the used hyperparameters for our method. Table 2.4, 2.5, 2.6, 2.7 and 2.8 provide the used hyperparameters for the baseline methods GACL, ME-ICRL and VME-ICRL respectively.

Table 2.2: Overview of the used hyperparameters for MCE-ICRL (part 1)

	Ant	Cheetah	Swimmer	Walker	Pendulum	Gridworld	HighD dist.	HighD vel.
General								
α	0	0	0	0	0	0	0	0
β	0.0001	0.0001	0.0001	0.0001	0.0001	0.00001	0.0001	0.0001
γ	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
Iterations	20	30	100	50	200	20	100	50
Expert trajectories	20	10	50	50	50	50	20	50
Max. trajectory length	500	1000	500	500	100	200	1000	1000
PPO								
Batch size	128	128	128	64	64	64	64	64
Learning rate	3.00E-05	3.00E-05	3.00E-05	0.0003	0.0001	3.00E-05	0.0001	0.0001
Steps	2048	2048	2048	2048	2048	2048	1024	1024
Epochs	20	20	20	10	20	10	20	20
Timesteps	200000	200000	50000	200000	100000	50000	50000	25000
GAE-λ	0.9	0.9	0.9	0.95	0.9	0.9	0.9	0.9
Target KL	0.02	0.02	0.02	0.01	0.01	0.01	0.02	0.02
Bootstrap timesteps	200000	200000	200000	200000	100000	35000	20000	0
Policy network	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64,64]	[64, 64]	[64, 64]

Table 2.3: Overview of the used hyperparameters for MCE-ICRL (part 2)

	Ant	Cheetah	Swimmer	Walker	Pendulum	Gridworld	HighD dist.	HighD vel.
λ, ζ								
Learning rate decay factor ζ	0.9	0.9	0.9	1	0.9	1	0.9	0.9
Batch size	64	64	64	64	128	64	1000	1000
Network ϕ_ζ	[40]	[40]	[40]	[40]	[64]	[40, 40]	[20]	[40, 20, 10]
Feature dimensions	64	64	64	64	16	64	10	8
Learning rate ζ	0.0003	0.0003	0.0003	0.0003	0.0005	0.0005	0.0001	1.00E-06
Learning rate λ	0.0001	0.0001	0.0001	0.0001	0.0001	0.0005	0.0001	1.00E-06
Bootstrap iterations	20	200	200	200	5	20	20	0
λ initial values	1	1	1	1	1	1	1	1

Table 2.4: Overview of the used hyperparameters for GACL

	Ant	Cheetah	Swimmer	Walker	Pendulum	Gridworld	HighD dist.	HighD vel.
Policy								
Batch size	128	64	64	64	128	64	64	64
Target KL	0.02	0.01	0.01	0.01	0.02	0.01	0.01	0.01
Learning rate	3.00E-05	0.0003	0.0003	0.0003	3.00E-05	0.0003	0.0005	0.0003
GAE-γ	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
GAE-λ	0.9	0.95	0.95	0.95	0.9	0.95	0.95	0.95
Policy network	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64,64]	[64, 64]	[64, 64]
Value network	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64,64]	[64, 64]	[64, 64]
Discriminator								
Architecture	[40, 40]	[30]	[30]	[64, 64]	[40, 40]	[40,40]	[30]	[30]
Learning rate	0.005	0.003	0.003	0.003	0.00001	0.003	0.003	0.003

Table 2.5: Overview of the used hyperparameters for ME-ICRL (part 1)

	Ant	Cheetah	Swimmer	Walker	Pendulum	Gridworld	HighD dist.	HighD vel.
Policy								
Batch size	128	64	128	128	128	64	64	64
Target KL	0.02	0.01	0.02	0.02	0.02	0.01	0.01	0.01
Learning rate	3.00E-05	3.00E-05	3.00E-05	0.0001	3.00E-05	0.0003	0.0001	0.0001
Timesteps	200000	200000	50000	200000	100000	50000	5000	5000
Reward-GAE-γ	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
Reward-GAE-λ	0.9	0.95	0.9	0.9	0.9	0.95	0.95	0.95
Cost-GAE-γ	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
Cost-GAE-λ	0.9	0.95	0.9	0.9	0.9	0.95	0.95	0.95
Policy network	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64,64]	[64, 64]	[64, 64]
Value network	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64,64]	[64, 64]	[64, 64]
Cost value network	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64,64]	[64, 64]	[64, 64]

Table 2.6: Overview of the used hyperparameters for ME-ICRL (part 2)

	Ant	Cheetah	Swimmer	Walker	Pendulum	Gridworld	HighD dist.	HighD vel.
Lagrangian λ								
Initial values	0.1	1	0.1	0.1	0.1	1	1	1
Learning rate	0.05	0.1	0.05	0.05	0.01	0.1	0.01	0.01
Budget λ	0	0	0	0	0	0	0	0
Constraint function,								
Architecture	[40, 40]	[20]	[40,40]	[64, 64]	[20]	[20,20]	[20]	[20]
Learning rate	0.005	0.05	0.005	0.001	0.001	0.05	0.0005	0.005
Backward iterations	5	10	5	5	5	10	10	10
Regularizer weight	0.6	0.5	0.6	0.6	0.6	0.5	0.5	0.5
Max forward KL	10	10	10	10	10	10	10	10
Max backward KL	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5

Table 2.7: Overview of the used hyperparameters for VME-ICRL (part 1)

	Ant	Cheetah	Swimmer	Walker	Pendulum	Gridworld	HighD dist.	HighD vel.
β-distribution								
α	0.9	0.9	0.9	0.009	0.09	0.9	0.9	0.9
β	0.1	0.1	0.1	0.001	0.01	0.1	0.1	0.1
Policy								
Batch size	128	64	128	128	128	64	64	64
Target KL	0.02	0.01	0.01	0.02	0.02	0.01	0.01	0.01
Learning rate	3.00E-05	3.00E-05	0.0003	0.0001	3.00E-05	0.0003	0.0001	0.0001
Timesteps	200000	200000	50000	200000	100000	50000	5000	5000
Reward-GAE-γ	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
Reward-GAE-λ	0.9	0.95	0.9	0.9	0.9	0.95	0.95	0.95
Cost-GAE-γ	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
Cost-GAE-λ	0.9	0.95	0.95	0.9	0.9	0.95	0.95	0.95
Policy network	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64,64]	[64, 64]	[64, 64]
Value network	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64,64]	[64, 64]	[64, 64]
Cost value network	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64, 64]	[64,64]	[64, 64]	[64, 64]

Table 2.8: Overview of the used hyperparameters for VME-ICRL (part 2)

	Ant	Cheetah	Swimmer	Walker	Pendulum	Gridworld	HighD dist.	HighD vel.
Lagrangian λ								
Initial values	0.1	1	5	0.1	0.1	1	1	1
Learning rate	0.05	0.1	0.01	0.05	0.01	0.1	0.01	0.01
Budget λ	0	0	0	0	0	0	0	0
Constraint function,								
Architecture	[40, 40]	[20]	[20]	[64, 64]	[20]	[20,20]	[20]	[20]
Learning rate	0.005	0.005	0.001	0.005	0.0001	0.05	0.0005	0.005
Backward iterations	5	10	5	5	5	10	10	10
Regularizer weight	0.6	0.5	0.6	0.6	0.6	0.5	0.5	0.5
Max forward KL	10	10	10	10	10	10	10	10
Max backward KL	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5

2.7.6 Additional Experimental Results

2.7.6.1 Gridworld

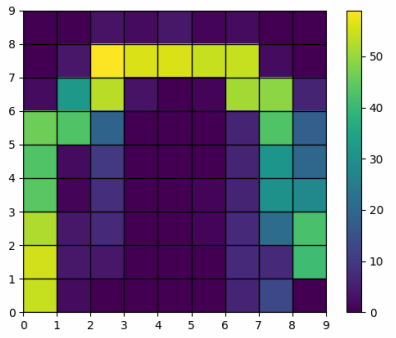
Figure 2.8(a) shows the states visited by the expert which are used as input for our method. Figure 2.8(b) reports the learned cost function when using MCE-ICRL with $\beta = 0.01$. Figure 2.8(c) shows the state visitations of the nominal agent. Figure 2.11 depicts the reward and constraint violation rate during training at different timesteps for the different methods for different rates of stochasticity. Figure 2.12 depicts the reward and constraint violation rate during training at different timesteps for our method with various value of β and for different rates of stochasticity. Figure 2.13 depicts the reward received and the constraint violation rate during training for the ablation study on the pre-training of the feature encoder.

2.7.6.2 Virtual Robotics Environments

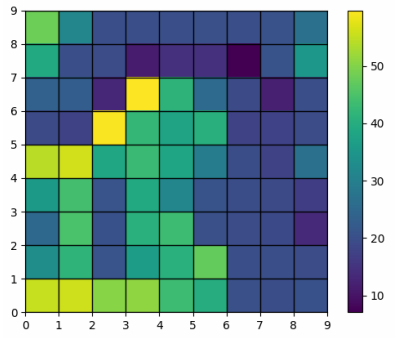
Figure 2.14 depicts the reward received and the constraint violation rate during training for different values of β. Figure 2.15 depicts the reward received and the constraint violation rate during training for the ablation study on the pre-training of the feature encoder.

2.7.6.3 Realistic Traffic Environment

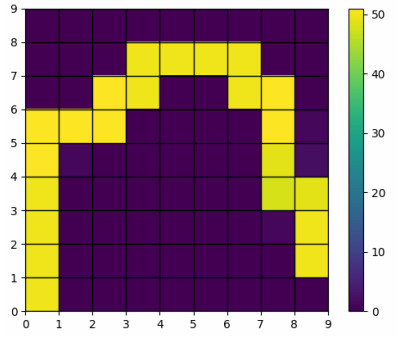
Figure 2.9 depicts the reward and constraint violation rate during training at different timesteps for our method with various value of β. Figure 2.10 depicts the reward received and the constraint violation rate during training for the ablation study on the pre-training of the feature encoder.



(a) Expert trajectories state visitations



(b) Recovered cost function



(c) State visitations for trajectories sampled from the nominal policy

Figure 2.8: Results from MCE-ICRL ($\beta = 0.01$) in the proposed gridworld environment.

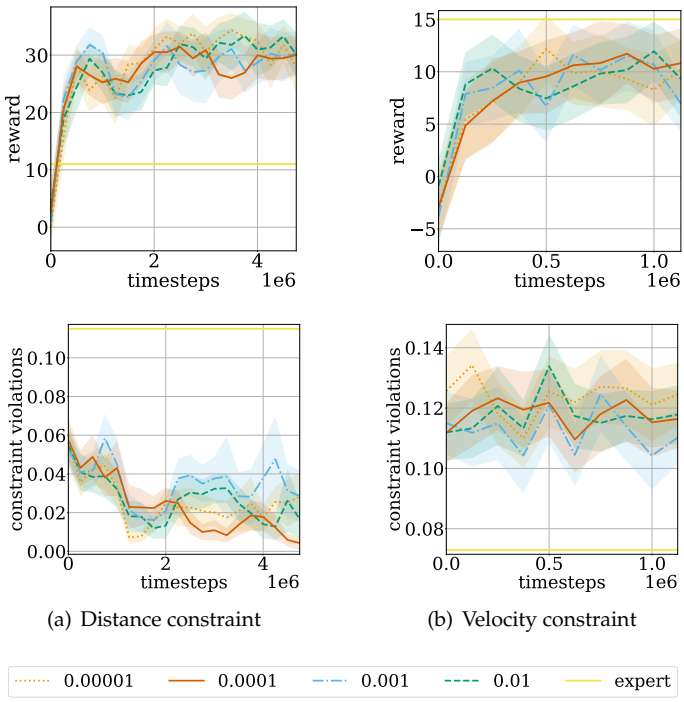
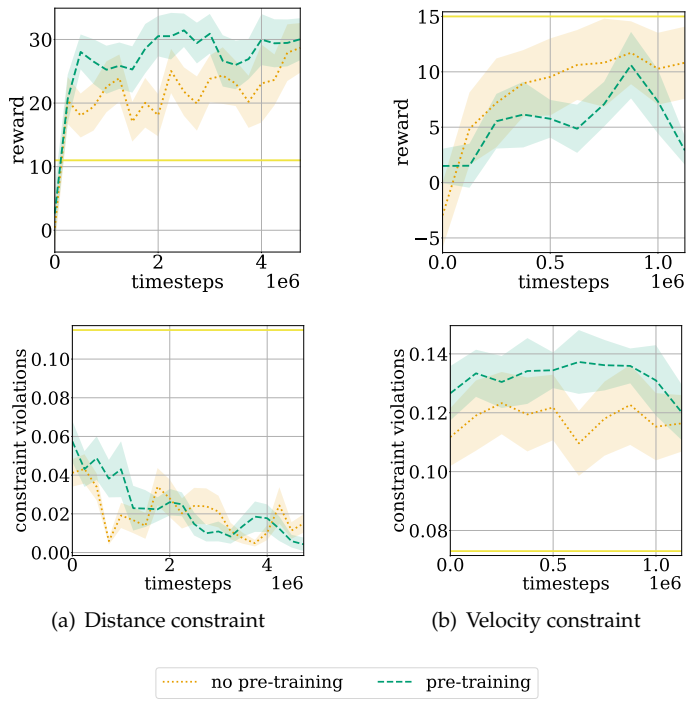


Figure 2.9: Evaluation of our method in the realistic traffic environment for different values of β: reward (top) and constraint violation rate (bottom) of trajectories sampled from the nominal policy during training. Results are averaged over 10 random seeds. The x-axis is the number of timesteps taken in the environment during training. The shaded regions correspond with the standard error.



(a) Distance constraint (b) Velocity constraint

Figure 2.10: Evaluation of our method in the realistic traffic environment for feature encoder pre-training disabled: reward and constraint violation rate of trajectories sampled from the nominal policy during training. Results are averaged over 10 random seeds. The x-axis is the number of timesteps taken in the environment during training. The shaded regions correspond with the standard error.

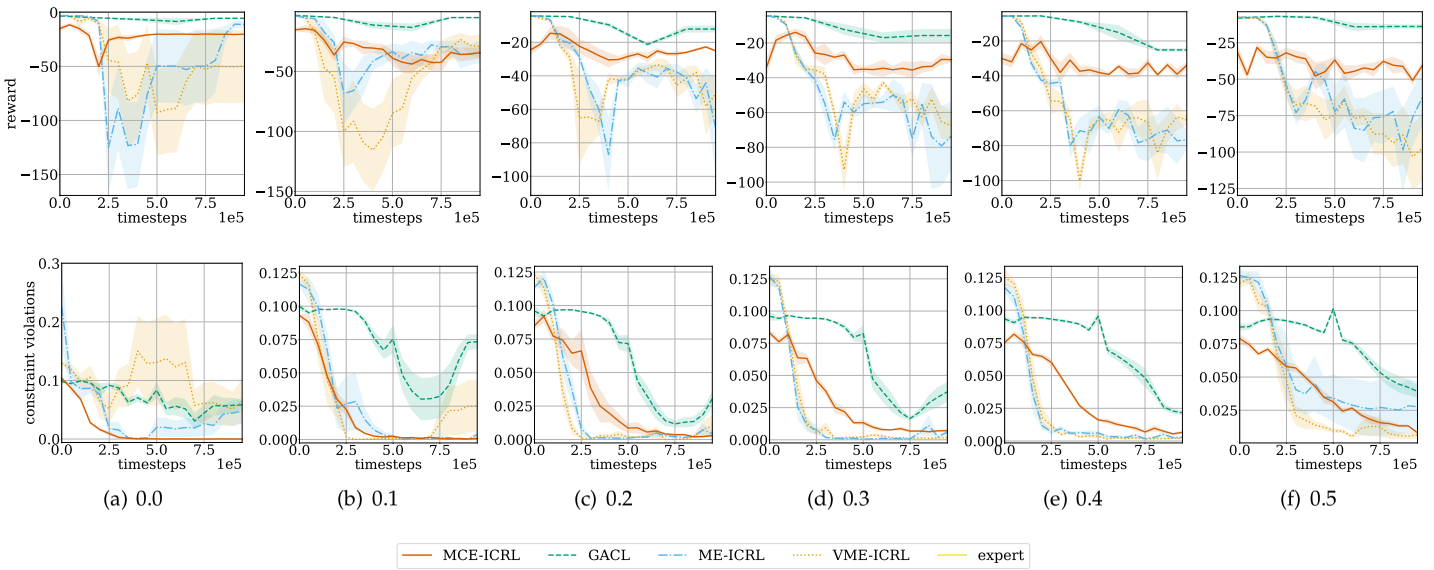


Figure 2.11: Evaluation of the different methods in the gridword environment for different rates of stochasticity $(0.0, 0.1, 0.2, 0.3, 0.4, 0.5)$: reward (top) and constraint violation rate (bottom) of trajectories sampled from the nominal policy during training. Results are averaged over 5 random seeds. The x-axis is the number of timesteps taken in the environment during training. The shaded regions correspond with the standard error.

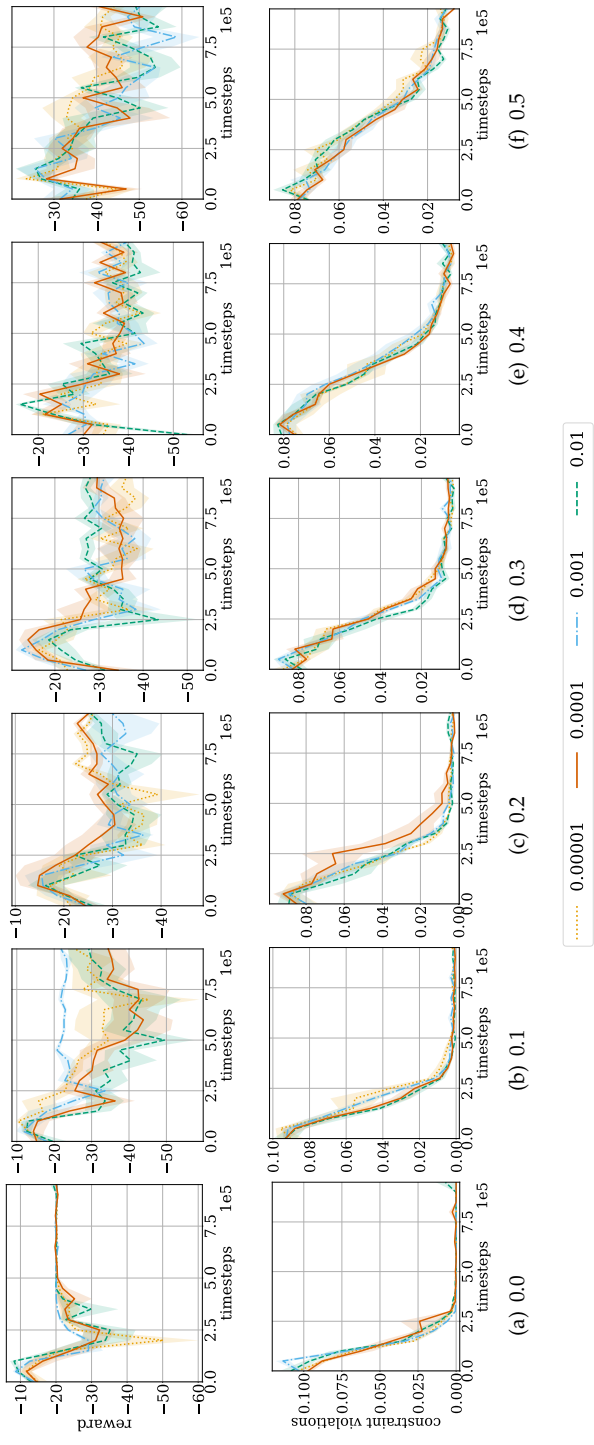


Figure 2.12: Evaluation of the different methods in the gridword environment for different rates of stochasticity $(0.0, 0.1, 0.2, 0.3, 0.4, 0.5)$ and different values of β $(0.00001, 0.0001, 0.001, 0.01)$: reward and constraint violation rate of trajectories sampled from the nominal policy during training. Results are averaged over 5 random seeds. The x-axis is the number of timesteps taken in the environment during training. The shaded regions correspond with the standard error.

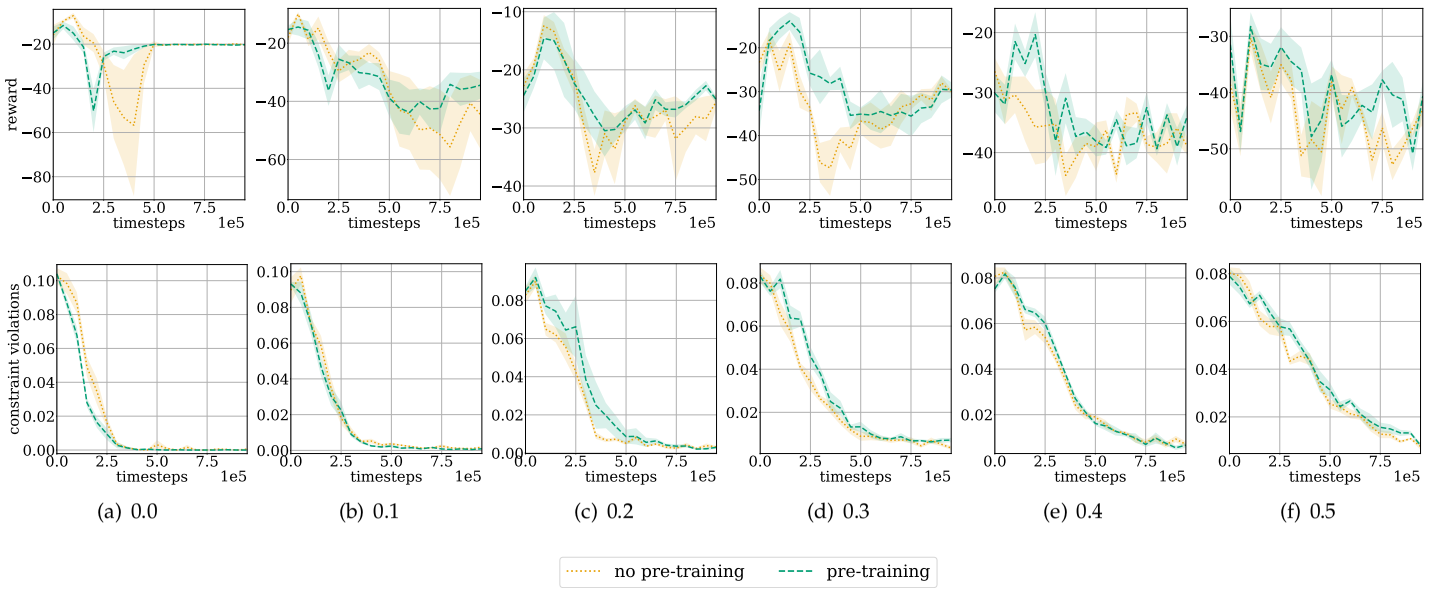
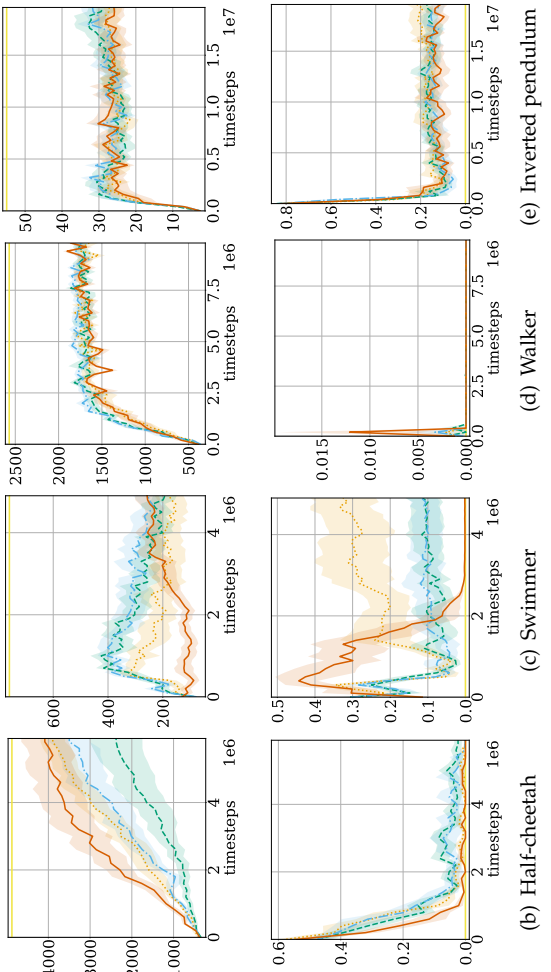
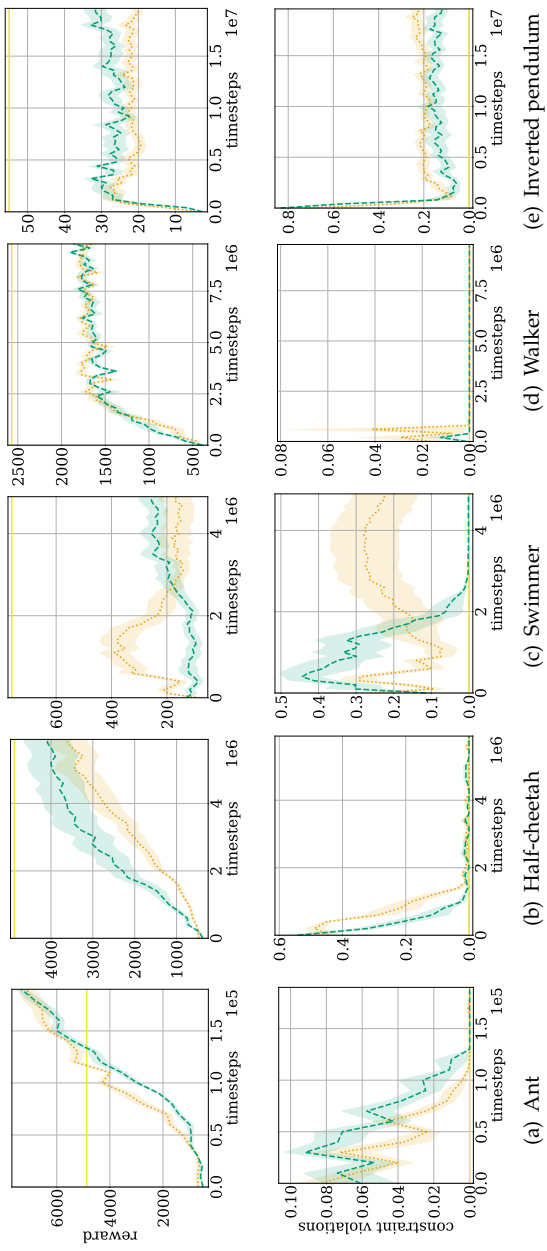


Figure 2.13: Ablation study on feature encoder pre-training in the gridworld environment for different rates of stochasticity $(0.0, 0.1, 0.2, 0.3, 0.4, 0.5)$: reward and constraint violation rate of trajectories sampled from the nominal policy during training. Results are averaged over 5 random seeds. The x-axis is the number of timesteps taken in the environment during training. The shaded regions correspond with the standard error.



(a) Ant (b) Half-cheetah (c) Swimmer (d) Walker (e) Inverted pendulum

Figure 2.14: Evaluation of our method in the virtual robotics environments for different values of β: reward and constraint violation rate of trajectories sampled from the nominal policy during training. Results are averaged over 10 random seeds. The x-axis is the number of timesteps taken in the environment during training. The shaded regions correspond with the standard error.



(a) Ant (b) Half-cheetah (c) Swimmer (d) Walker (e) Inverted pendulum

Figure 2.15: Ablation study on feature encoder pre-training in the virtual robotics environments: reward and constraint violation rate of trajectories sampled from the nominal policy during training. Results are averaged over 10 random seeds. The x-axis is the number of timesteps taken in the environment during training. The shaded regions correspond with the standard error.

3

Logic Constraint Inference

"No eternal reward will forgive us now for wasting the dawn."
The Doors, "Stoned Immaculate"

Inverse Reinforcement Learning Through Logic Constraint Inference

M. Baert • S. Leroux • P. Simoens.
Published in Machine Learning 112, 2023

In the previous chapter, we introduced an approach to scale ICRL to complex environments with stochastic transition dynamics. In this chapter, we shift our focus to the interpretability of constraint representations in ICRL. To achieve this, we integrate a rule induction engine into the iterative process of constraint inference and policy refinement. Each iteration, the system generates first-order logic formulas (i.e., rules) that abstract and generalize the identified constraints, providing a more interpretable and structured representation. This enhancement brings two key advantages. First, by encapsulating multiple states within a single logical rule, the constraint inference process becomes more efficient, reducing the number of iterations required. Second, the interpretability of logical rules offers clear insights into the learned constraints, facilitating validation and understanding. We demonstrate the effectiveness of our method through experiments in two environments: the Tower of Hanoi and a traffic simulation. In both cases, our approach successfully induces a compact and interpretable logic program that captures the behavioral constraints governing the respective tasks.

3.1 Introduction

Over the last few years, the application domain of robots has expanded from installations in caged factory environments to deployments in environments shared with humans. The behavior of humans in these environments is typically guided by explicit (rules) and implicit regulations. We jointly refer to both types of regulations as social norms which represent shared standards of acceptable behavior [1]. Asides from reaching a goal, all (artificial) agents should respect the applicable social norms to assure safety and predictability [2].

Reinforcement learning (RL) is a general framework for decision-making and control. It is a training method for learning a mapping from states to actions (i.e., a policy) in order to maximize a predefined reward signal. However, fully defining the desired behavior of an artificial agent by a single reward function is often a burdensome task [3, 4, 5]. The framework of constrained reinforcement learning specifies an agent's desired behavior by a reward function in combination with a set of hard constraints. For instance, the reward function can be expressed in terms of the distance from the goal and the time needed to reach it, complemented with a hard constraint to

not crash into other vehicles. In the Markov decision process representing the environment, constraints are states or state-action pairs that in principle could be reached by a certain trajectory, but that are considered as invalid by latent information about the environment, such as social norms. In contrast to reward functions, hard constraints provide strong safety guarantees such that a constrained state will never be visited, this makes them better suited for decision-making in safety-critical environments. Because social norms are often implicit and environment-specific, it is complicated to define them as a set of constraints. A promising direction is to learn social norms from human observations. To this end, recent studies have shown principles of inverse reinforcement learning [6] can be adopted to infer both reward function and constraints from observed trajectories [7, 8].

A shortcoming of current methods is the lack of interpretability and hence verifiability of what an agent has actually learned. When learning from human observations in the wild, there is a chance that the dataset will contain trajectories of humans violating to a smaller or larger extent the social norms, e.g., tailgating, accelerating when the traffic light turns orange, etc. To guarantee safety, we should be able to validate which aspects were extracted from the observations since we want to refrain an agent from picking up undesired behavior in the dataset. Although a single constraint, represented by a state-action pair, could be easily interpretable, it is difficult to obtain a general view on the environmental restrictions from a large set of constrained state-action pairs. Another limitation is the difficulty to transfer learned constraints in one environment to a similar environment, since constraints are defined in the state-action space of a particular environment. We hypothesize that defining constraints in terms of more generalized, human interpretable concepts would improve transferability. For example, when inferring constraints from a traffic environment, all states corresponding to off-road positions should be added to the set of constraints. However, when humans learn to drive, we use the concept "road" and learn a single rule which prohibits us to drive off the road. In contrast with the set of constraints defined as prohibited state-action pairs, this rule also applies in traffic environments other than the one which was used to learn the rule.

Inductive logic programming (ILP) [9] offers a computational framework to induce a logic program (i.e., a hypothesis) which generalizes a set of training examples. ILP also provides the possibility to specify background knowledge as a logic program. Because the induced hypothesis is comprehensible by humans and providing background knowledge allows us to introduce human concepts, ILP could be a worthy alternative for IRL and constraint inference. However, while ILP can, in principle, learn from only

positive examples, it typically relies on both positive and negative examples to constrain the hypothesis space. Learning from solely positive examples is considerably more challenging, as it increases the risk of overgeneralization and demands stronger inductive biases. Negative examples correspond to anomalous behavior which is rarely observed in real-world environments or would be dangerous to collect. If the observations contain anomalous behavior, this will be reflected in the induced rules hence can be detected.

In this chapter, we combine ILP and constraint inference and propose a hybrid method to infer restrictive rules (which represent social norms) from observations (positive examples) and represent them by a logic program. The presented method is an iterative procedure, where each iteration consists of an inference and an induction stage. During inference, a constraint is inferred from the observed trajectories and added to a set of constraints. During induction, a logic program (hypothesis) is induced which generalizes the observed trajectories (positive examples) and the set of inferred constraints (negative examples). The hypothesis is then translated back to the state-action space and used to update the model of the environment for the next iteration. Since the learned program is expressed in formal logic, we can interpret and validate which aspects of the observed behavior are extracted from the observations, and adjust the induced hypothesis if necessary. We show our method is able to induce a compact set of rules which can be used to accurately classify behavior to be valid or invalid. We provide empirical evidence that the learned set of rules can be provided as prior knowledge to a model-free RL agent. This reduces the required number of episodes until convergence and prevents any social norm violation both during training and deployment. An ablation study demonstrates the importance of the induction step. Furthermore, experimental results confirm that expressing constraints in a symbolic language facilitates knowledge transfer to similar environments.

The remainder of this article is structured as follows. Section 3.2 introduces the foundations of this work: (constrained) Markov decision processes, inverse reinforcement learning, answer set programming and inductive learning of answer set programs. Section 3.3 elaborates on our proposed method. The proposed method and the induced hypotheses are tested and evaluated in section 3.4. Section 3.5 provides an overview of related work. Concluding remarks and future directives are discussed in section 3.6.

3.2 Background

3.2.1 Constrained Markov Decision Process

A finite-state Markov decision process (MDP) is a model for sequential decision-making defined as $\mathcal{M} = (S, \{A_s \mid s \in S\}, \mathcal{G}, P_{\text{trans}}, P_0, R, \phi)$ [10]. S is a finite set of discrete states and A a finite set of discrete actions. $\{A_s \mid s \in S\}$ denotes the set of sets of available actions for a given state $s \in S$, such that $A_s \subseteq A$. $\mathcal{G} \subset S$ represents the set of goals as a subset of all states. $P_{\text{trans}} : S \times A \times S \mapsto [0, 1]$ denotes the transition probability distribution where $P_{\text{trans}}(s' \mid s, a)$ expresses the probability of transitioning to state s' when performing action a while in state s. $P_0 : S \times \mathcal{G} \mapsto [0, 1]$ denotes the probability distribution over initial states for each goal. $R : S \times A \times S \times \mathcal{G} \mapsto \mathbb{R}$ is a goal-dependent reward function where $R(s, a, s', g)$ denotes the scalar reward the agent receives for taking action a while in state s, transitioning to state s' and aiming for goal g. We consider an agent interacting with the environment at discrete time steps t generating a sequence of transitions called a trajectory $\tau = ((s_1, a_1, s_2), ..., (s_{T-1}, a_{T-1}, s_T))$ of length T. At every time step, the agent selects its action based on a goal-conditioned policy $\pi : S \times \mathcal{G} \mapsto A$. Solving the MDP corresponds to finding the optimal policy π^* which maximizes the expected total discounted reward for all goals $g \in \mathcal{G}$:

$$J(\pi) = \mathbb{E}_{\tau \sim \pi} \sum_{t=0}^{T} \gamma^t R(s_t, a_t, s_{t+1}, g) \tag{3.1}$$

with $\gamma \in [0, 1]$. We define a constrained Markov decision process (CMDP) $\mathcal{M}^C$ as a nominal (i.e., unconstrained) MDP $\mathcal{M}$ imposed with a set of constraints $C \subseteq S \times A$. We impose a set of constraints C on $\mathcal{M}$ by restricting the set of valid actions in each state A_s. The set of valid actions in state s in the CMDP $\mathcal{M}^C$ is defined as: $A_s^C = A_s \setminus \{a \in A_s \mid (s, a) \in C\}$. It is possible that due to the imposed constraints, for some states the set of valid actions becomes empty $A_s = \emptyset$. Such states are referred to as empty states and the set of empty states is denoted by S_{empty}.

3.2.2 Inverse Reinforcement Learning

Inverse reinforcement learning (IRL) takes as input an MDP without reward function $\mathcal{M} \backslash R$ and a set of demonstrated trajectories $\mathcal{T} = (\tau_1, ..., \tau_N)$ sampled from an unknown expert policy π_E. $\phi : S \times A \mapsto \mathbb{R}^k$ defines a function which maps a state-action pair to a k-dimensional feature space where $\phi(s, a)$ denotes the feature vector representing state-action pair (s, a). $\phi(\tau) = \sum_{t=1}^{T-1} \phi(s_t, a_t)$ defines the feature representation of a trajectory as the sum of the features of all state-action pairs in that trajectory. The goal of IRL

is to learn a reward function $\hat{R}$ which best explains the observed behavior. Feature expectation matching provides a general method for solving the IRL problem by matching the expected features of the trajectories obtained by optimizing the recovered reward with the features of the trajectories provided by the expert: $\min_{\hat{R}}[\mathbb{E}(\phi(\tau) \mid \pi_L) - \mathbb{E}(\phi(\tau) \mid \pi_E)]$[11]. Here π_L denotes the learner's policy which is implied by the learner's reward $\hat{R}$, the expert policy π_E is reflected by the expert's trajectories $\mathcal{T}$. However, the problem of IRL and expected feature matching is ill-posed since many reward functions in the set of all reward functions can explain a finite set of suboptimal demonstrations, thus induce the same expected features. Maximum entropy (MaxEnt) IRL [12] provides a general probabilistic framework to resolve this ambiguity. The expected features are expressed as a sum over trajectories: $\mathbb{E}(\phi(\tau) \mid \pi_L) = \sum_{\tau} P_{\pi_L}(\tau)\phi(\tau)$ with $P_{\pi_L}(\tau)$ the probability of trajectory τ under the learner's policy π_L. The probability of a trajectory is assumed to be exponentially proportional to the reward earned by that trajectory: $P_\pi(\tau) \propto e^{R(\tau)}$. Then, the MaxEnt principle proposes to choose the distribution with minimal bias by selecting the trajectory distribution $P_{\pi_L}(\tau)$ with maximum entropy which matches features with the expert's trajectories.

3.2.3 Answer Set Programming

Answer set programming (ASP) is a declarative problem-solving approach [13]. We present a restricted definition of ASP that highlights the aspects relevant to this chapter. For a comprehensive overview, please refer to [14]. In ASP a normal rule r is defined as h :- $b_1, ..., b_m$, `not` $b_{m+1}, ...,$ `not` b_n. The head of the rule head(r) is an atom h and b_i are literals which constitute the body of the rule. We define body$^+(r) = \{b_1, ..., b_m\}$ and body$^-(r) = \{b_{m+1}, ..., b_n\}$. The rule should be read as: if all atoms in body$^+(r)$ are true and all atoms in body$^-(r)$ cannot be proven to be true, then h should be true. Body-less rules are referred to as facts, in this case the :- sign will be dropped. An integrity constraint is defined as a headless rule: :- $b_1, ..., b_n$. This means that no solution can satisfy simultaneously b_1 and b_2 and ..., b_n. A choice rule is of the form $l\{h_1; ...; h_n\}u$:- $b_1, ..., b_n$ with l and u being integers such that $0 \leqslant l \leqslant u \leqslant n$. At least l and at most u terms of $\{h_1, ..., h_n\}$ can be true if the body of the rule is satisfied. The body of integrity constraints and choice rules can also contain negated literals. An answer set program P is a finite set of normal rules, integrity constraints and choice rules. Given P, the Herbrand base B_π is the set of all ground (variable-free) atoms constructed from predicate names and constant symbols that occur in P. The Herbrand interpretations of P correspond to the set of interpretations which cover all possible combinations of atom assignments.

To solve an answer set program P, the program is first transformed to a logic program without variables (grounding). An interpretation X is a model of a propositional logic program, if $\text{head}(r) \in X$ whenever $\text{body}^+(r) \subseteq X$ and $\text{body}^-(r) \cap X = \emptyset$ for every $r \in P$. However in ASP, the semantics of a program are given by its *stable models*. An interpretation X is a stable model of P, if X is the $\subseteq$-smallest model of the reduct P^X. We use the definition of the *simplified* reduct for a restricted class of programs containing only normal rules, choice rules and hard constraints introduced by [15]. The *simplified* reduct of a ground program P with respect to some interpretation X, is constructed following four sequential steps:

- Remove any rule r for which $\text{body}^-(r) \cap X \neq \emptyset$.
- For any constraint r, :-body(r), replace r with $\bot$:-body$^+(r)$ ($\bot$ is a new atom which cannot appear in any answer set of P).
- For any choice rule r, $l\{h_1; ...; h_n\}u$:-body(r) such that $l \leq| X \cap \{h_1; ...; h_n\} |\leq u$, replace r with the set of rules $\{h_i$:-body$^+(r) \mid h_i \in X \cap \{h_1, ..., h_n\}\}$.
- For any remaining choice rule r, replace r with the constraint $\bot$:-body$^+(r)$.

In this work, we adopt clingo [16] to ground and solve ASP programs.

3.2.4 Inductive Learning of Answer Set Programs

Inductive learning of answer set programs (ILASP) [15] is a system for learning an ASP program, referred to as a *hypothesis*, from examples of what should and what should not be an answer set. A partial interpretation e is a pair of sets of atoms $(e^{\text{inc}}, e^{\text{exc}})$ where e^{inc} represents the atoms which are true and e^{inc} the atoms which are false. A Herbrand interpretation X extends a partial interpretation e iff $e^{\text{inc}} \subseteq X \wedge e^{\text{exc}} \cap X = \emptyset$. The input of the learning task consists of an ASP program B representing background knowledge, two partial interpretations denoted by the positive and negative examples E^+ and E^- and a hypothesis space S_M defined by a language bias M. The goal of ILASP is to find a hypothesis H such that:

[1] $H \subseteq S_M$.

[2] $\forall e^+ \in E^+ : \exists A \in AS(B \cup H)$ s.t. A extends e^+.

[3] $\forall e^- \in E^- : \nexists A \in AS(B \cup H)$ s.t. A extends e^-.

Here, A represents a single answer set and $AS(P)$ depicts the set of all answer sets of a logic program P. The first condition states that the hypotheses H is composed of rules from the hypothesis space S_M. The second condition declares the existence, for every positive sample, of at least one answer

set of $B \cup H$ which extends that example. The third condition assures no answer set of $B \cup H$ extends any of the negative examples. If H meets all three conditions, H is an inductive solution of the learning task defined by the 4-tuple: (S_M, B, E^+, E^-).

3.3 Method

In this section we formally describe the proposed method. Figure 3.1 shows a single iteration of the algorithm applied to the towers of Hanoi domain. The input consists of a nominal MDP $\mathcal{M}$, a set of trajectories $\mathcal{T}$ in a constrained MDP $\mathcal{M}^{\hat{C}}$ with $\hat{C}$ the ground truth set of constraints which is unknown, an ASP program B representing background knowledge about the environment and a hypothesis space S_M defined by a language bias M. The goal of our method is to induce a hypothesis H, represented by an ASP program, which covers all constraints in $\hat{C}$. Because we consider a finite set of trajectories, a state-action pair which is not observed can still be valid. We assume no invalid state-action pairs occur in any trajectory. We propose an iterative procedure where each iteration consists of a constraint inference and program induction stage. The number of iterations η is a hyperparameter. First, the optimal policy is learned in the nominal (unconstrained) MDP $\mathcal{M}$. From this nominal policy and the set of expert trajectories, one state-action pair is selected by the principle of maximum likelihood [7] and added to the set of constraints C. This is the state which has the highest likelihood under the nominal policy but does not occur in any expert trajectory. Thus, a state which is not visited by an expert but will likely be visited by an agent which is not aware of the implicit rules in the environment. In the towers of Hanoi domain, the nominal policy will induce behavior which moves the pile from the initial position to the goal position without taking the rules into account. The optimal nominal policy will first move disk 2 to the middle peg, move disk 1 to the middle peg (on disk 2), move disk 0 to the right peg (this state corresponds to the inferred constraint which is visualized in Figure 3.1 because this state has the highest likelihood under the nominal policy and does not occur in any of the expert trajectories), move disk 1 to the right peg and finally move disk 2 to the right peg. Next, ILASP induces a hypothesis from the set of constraints, which serves as the set of negative examples, and the set of trajectories, which serves as the set of positive examples. The set of constraints is updated by translating the hypothesis back to state-action space, i.e., all state-action pairs which are invalidated by the induced hypothesis are added to the set of constraints. Finally, the updated set of constraints is augmented on the MDP by restricting the available actions in each state $\{A_s\}$. The next iteration makes use of this constrained version of the nominal MDP.

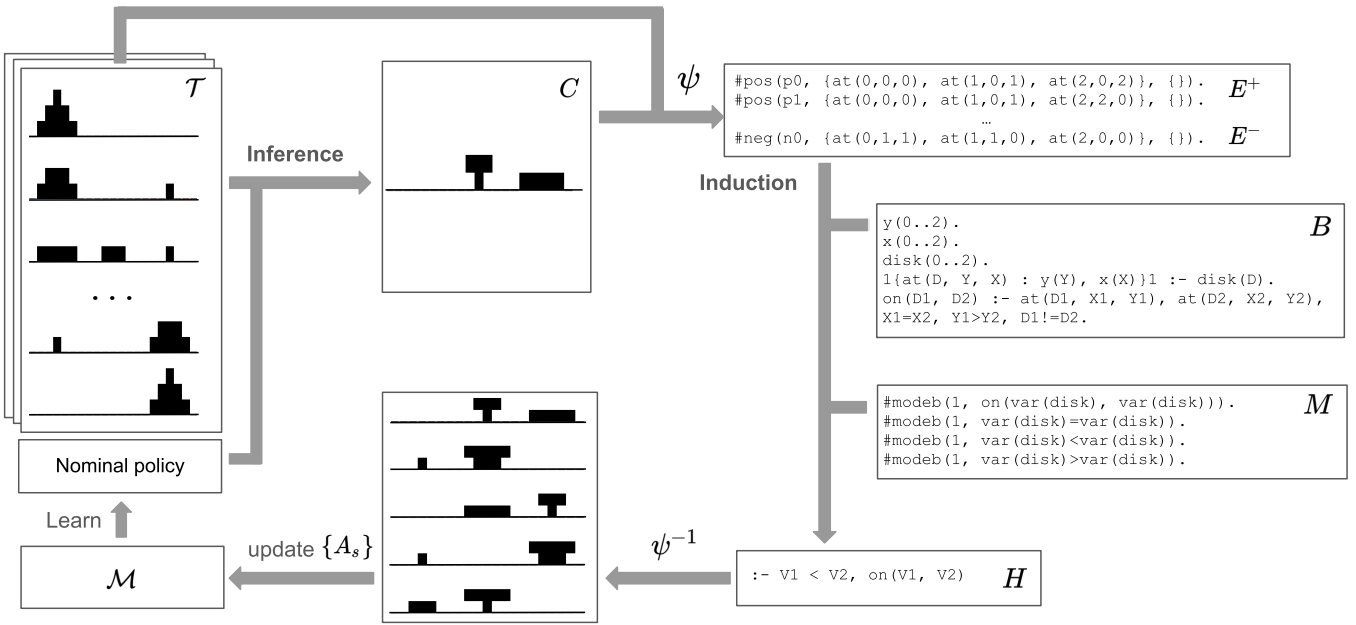


Figure 3.1: Overview of a single iteration of the proposed method applied to the towers of Hanoi environment. First, an optimal policy is learned from the nominal MDP $\mathcal{M}$. During the inference step, a constraint is inferred from a set of trajectories $\mathcal{T}$ and the nominal policy. The constraint is added to the set of constraints C. The set of constraints and trajectories are mapped to positive E^+ and negative examples E^- using ψ. From the examples, a hypothesis H is induced using ILASP given background knowledge B and language bias M (induction step). The answer sets of $H \cup B$ are translated back to state-action space with ψ^{-1} and are used to update the set of constraints C. At last, the set of constraints is augmented on $\mathcal{M}$ by updating the set of available actions in each state $\{A_s\}$.

Subsection 3.3.1 provides a formal description on the constraint inference stage, i.e., how the most likely constraint is selected using the observed trajectories. Subsection 3.3.2 presents the program induction stage and the integration with constraint inference. The complete algorithm is depicted in Algorithm 3.1.

3.3.1 Constraint Inference

Since the set of observed trajectories is finite, the absence of a state-action pair in the observations does not necessarily imply that this pair is invalid. To infer the state-action pairs which are most likely invalid, we build on the principle of maximum likelihood constraint inference [7]. More formally, Scobee et al. propose an iterative process where each iteration a constraint $c^* \in \mathcal{C}$ is added to the set of constraints C. This is the constraint which, when augmented on the MDP, maximizes the likelihood of the observed trajectories. The constraint space is defined as all possible state-action pairs: $\mathcal{C} = S \times A$. We consider a CMDP $\mathcal{M}^C$ for which the set of constraints is initially empty. We define the probability of a set of trajectories $\mathcal{T}$ as $P_{\mathcal{M}^C}(\mathcal{T})$. The maximum likelihood objective is defined as:

$$c^* = \arg\max_{c \in \mathcal{C}} P_{\mathcal{M}^{C \cup c}}(\mathcal{T}). \tag{3.2}$$

Following the model of maximum entropy [12], the probability of a set of trajectories $\mathcal{T}$ is exponentially proportional to the reward earned by each trajectory $\tau \in \mathcal{T}$:

$$P_{\mathcal{M}^C}(\mathcal{T}) = \frac{1}{Z(C)^{|\mathcal{T}|}} \prod_{\tau \in \mathcal{T}} e^{R(\tau)} \mathbb{1}^{\mathcal{M}^C}(\tau). \tag{3.3}$$

The indicator $\mathbb{1}^{\mathcal{M}^C}(\tau)$ signifies if a trajectory τ is valid in $\mathcal{M}^C$. The partition function $Z(C)$ is defined as the sum of the exponentiated rewards over the set of all valid trajectories in $\mathcal{M}^C$, $\xi_{\mathcal{M}^C}$:

$$Z(C) = \sum_{\tau \in \xi_{\mathcal{M}^C}} e^{R(\tau)}. \tag{3.4}$$

To make $\xi_{\mathcal{M}^C}$ finite, only trajectories with length smaller or equal to the maximum planning horizon $\Gamma \in \mathbb{N}^+$ are considered. To solve equation (3.2), $Z(C)$ should be minimized while not invalidating any of the observed trajectories. Scobee et al. derive that this can be done by maximizing the sum of the exponentiated rewards over all trajectories made invalid by augmenting a constraint c on $\mathcal{M}^C$. This set of invalid trajectories is denoted by $\xi^-_{\mathcal{M}^{C \cup c}}$. Given this, the objective becomes:

$$\begin{aligned} c^* = \arg\max_{c \in \mathcal{C}} & P_{\mathcal{M}}(\xi^-_{\mathcal{M}^{C \cup c}}) \\ \text{s.t. } & \mathcal{T} \cap \xi^-_{\mathcal{M}^{C \cup c}} = \emptyset. \end{aligned} \tag{3.5}$$

This step comes down to adding the state-action pair to the set of constraints which is not observed in any of the given trajectories but has the highest probability under $\mathcal{M}^C$. To determine the probability of a state-action pair, we define the state-action visitation frequency:

$$D_{s',a'} = \sum_{g \in \mathcal{G}} D_{s',g} \pi(a' \mid s', g) \mathbb{1}^{\mathcal{M}^C}(s', a'). \tag{3.6}$$

Where $D_{s',g}$ denotes the single goal state visitation frequency which can be calculated recursively:

$$D_{s',g} = \sum_{s \in S} \sum_{a \in A_s} D_{s,g} \pi(a \mid s, g) P_{trans}(s' \mid s, a). \tag{3.7}$$

The final objective then becomes:

$$c^* = \arg\max_{s,a} D_{s,a} \;\text{ s.t. }\; (s, a) \notin \mathcal{T}. \tag{3.8}$$

The single goal state visitation frequency $D_{s',g}$ is initialized with the goal-conditioned initial state distribution P_0. Each iteration the goal-conditioned policy is calculated using the backward pass of the MaxEnt IRL algorithm [12]:

$$\pi(a \mid s, g) = \frac{Z_{s,a,g}}{Z_{s,g}} \tag{3.9}$$

with the state-action partition function $Z_{s,a,g}$ and the state partition function $Z_{s,g}$ defined recursively as:

$$Z_{s,a,g} = \sum_{s' \in S} Z_{s',g} P_{trans}(s' \mid s, a) e^{R(s,a,s',g)} \tag{3.10}$$

$$Z_{s,g} = \sum_{a \in A_s} Z_{s,a,g}. \tag{3.11}$$

Once a constraint c^* is inferred, it is added to the set of constraints C together with all empty states. To prevent transitioning to an empty state, state-action pairs (s, a) for which $P_{\text{trans}}(s' \mid s, a) > 0$ and $s' \in S_{\text{empty}}$ are also added to the set of constraints. Earlier work [8, 7] assumes the reward function, which is required to calculate the policy, is known or calculated using an IRL method. To relax this restriction, we propose to infer a sparse reward function from the observed trajectories such that the reward is 1 if the next state is a terminal state in one of the observed trajectories, otherwise the reward is zero.

$$R(s, a, s', g) = \begin{cases} 1 & \text{if } s' \in \mathcal{T}_T^g \\ 0 & \text{otherwise.} \end{cases} \tag{3.12}$$

Here $\mathcal{T}_T^g$ denotes the set of states at the final time step of all trajectories where the observed agent pursues goal g.

3.3.2 Program Induction

The goal of our work is to induce a hypothesis, expressed as an ASP program, which explains the behavior of agents in an environment. In this subsection we describe how constraint inference and ILASP can be integrated to generalize the set of inferred constraints to a set of rules expressed in formal logic. The input of ILASP consists of a set of positive examples E^+ and a set of negative examples E^-, an ASP program representing background knowledge B about the environment and a search space S_M. Both B and S_M are manually engineered. The set of positive examples originates from the observed trajectories $\mathcal{T}$ while the set of negative examples is based on the set of constraints C. However, ILASP requires all inputs are expressed as first-order logic predicates and facts while the observed trajectories and inferred constraints are defined in the MDP's state-action space. Because of this nonconformity, we define an environment-specific mapping ψ from the state-action space to first-order logic. Thus, the set of positive and negative examples can be formally defined as:

$$\psi(s,a) \in E^+ : (s,a) \in \mathcal{T} \tag{3.13}$$

$$\psi(s,a) \in E^- : (s,a) \in C. \tag{3.14}$$

An example of ψ for the towers of Hanoi environment can be found in Figure 3.1. Running ILASP results in a hypothesis H which possibly generalizes the given examples over a larger region of the state-action space. To benefit from this generalization in the next iteration, we translate the ASP program H back to the state-action space using the inverse mapping ψ^{-1} and update the set of constraints C accordingly. The set of constraints is extended with the state-action pairs which correspond to the interpretations which are an answer set of B but are no answer set of $B \cup H$, i.e., the state-action pairs which are no longer valid after imposing H on B. After η iterations, the hypothesis H is returned. The complete procedure is depicted in Algorithm 3.1.

3.4 Experiments

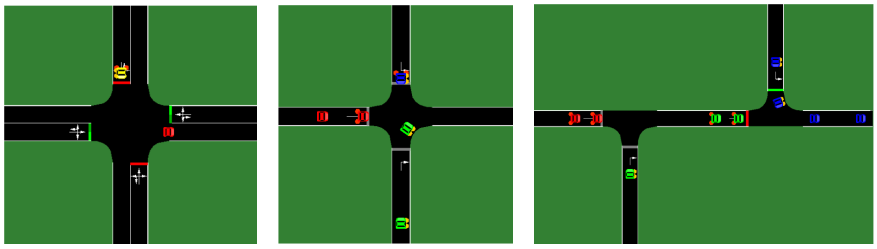
In this section we perform an experimental evaluation of the proposed method. Subsection 3.4.1 provides a brief overview of the used environments. In subsection 3.4.2, a qualitative evaluation follows of the induced hypotheses. Subsection 3.4.3 covers a quantitative evaluation of the learned rules and the effect of the hyperparameter η and the number of trajectories available. In subsection 3.4.4, we provide a model-free RL agent with rules induced using our method as prior knowledge. First, we examine to what extent this prior knowledge speeds up the learning process. Next, we evaluate the applicability in safety critical environments by measuring the

Algorithm 3.1: Logic constraint inference algorithm

Input: nominal MDP $\mathcal{M}$, expert trajectories $\mathcal{T}$, search space S_M, background knowledge B
Parameters: planning horizon Γ, number of iterations η
Output: hypothesis H

$C \leftarrow \emptyset$
for $i \leftarrow 0$ *to* η **do**
  `// Constraint inference`
  for $g \in \mathcal{G}$ **do**
    for $t \leftarrow 0$ *to* Γ **do**
      $Z_{s,a,g} \leftarrow \sum_{s' \in S} Z_{s',g} P_{trans}(s' \mid s, a) e^{R(s,a,s',g)}$
      $Z_{s,g} \leftarrow \sum_{a \in A_s} Z_{s,a,g}$
    end
    $\pi(a \mid s, g) \leftarrow \frac{Z_{s,a,g}}{Z_{s,g}}$
  end
  while *not converged* **do**
    $D_{s',g} \leftarrow \sum_{s \in S} \sum_{a \in A_s} D_{s,g} \pi(a \mid s, g) P_{trans}(s' \mid s, a)$
    $D_{s',a'} \leftarrow \sum_{g \in \mathcal{G}} D_{s',g} \pi(a' \mid s', g) \mathbb{1}^{\mathcal{M}^C}(s', a')$.
  end
  $c^* \leftarrow \arg\max_{s,a} D_{s,a}$ s.t. $(s, a) \notin \mathcal{T}$
  $C \leftarrow C \cup \{c^*\}$
  $C \leftarrow C \cup \{(s, a)\} : s \in S_{\text{empty}}, a \in A_s$
  $C \leftarrow C \cup \{(s, a)\} : P_{\text{trans}}(s' \mid s, a) > 0, s' \in S_{\text{empty}}$
  $E^+ \leftarrow \psi(\mathcal{T})$
  `// Program induction`
  $E^- \leftarrow \psi(C)$
  $H \leftarrow ILASP(S_M, B, E^+, E^-)$
  $C \leftarrow \psi^{-1}(\text{AS}(B) \setminus \text{AS}(B \cup H))$
  $A_s^C \leftarrow A_s \setminus \{a \in A_s \mid (s, a) \in C\}$
end
return H

empirical violation probability both during and after training. At last, in subsection 3.4.5 we inspect the transferability of the induced rules to other but similar environments.



(a) Traffic light controlled intersection

(b) Intersection with priority to the right

(c) Combination of 3.2(a) and 3.2(b)

Figure 3.2: The different SUMO traffic environments. The agents learn social norms such as dealing with priority to the right and traffic lights, purely from observed trajectories.

3.4.1 Environments

We conduct experiments in the towers of Hanoi environment (Figure 3.1) and simulated traffic environments (Figure 3.2). We chose these environments because the behavior of agents in these environments is constrained by formal or informal rules.

Towers of Hanoi is a canonical recursive problem where a stack of disks has to be moved from one place to another, without placing a larger disk on a smaller one and only moving the upper disk of any stack. Although all rules are explicitly known and could in principle be integrated upfront in the MDP definition, the simplicity of this problem allows illustrating the working of our algorithm. In this environment, there is only one agent which plays the game. The state of the game is represented by the xy-coordinates of the three disks. The available actions are the displacement of any disk on the top position of one of the stacks to the top position of another stack. To establish a mapping to first-order logic using ψ, we use three instances (one for each disk) of the predicate `at(D, X, Y)`. This predicate evaluates true if disk `D` is at position `(X, Y)`. Disks are numbered in order of decreasing size, the disk for which `D=1` corresponds with the largest disk.

For the second set of experiments, we constructed three traffic environments: an intersection with bidirectional traffic which is controlled by traffic lights (Fig 3.2(a)), an intersection with unidirectional traffic where the priority is given to cars coming from the right (Fig 3.2(b)) and an environment combining both previous scenarios (Fig 3.2(c)). In these environments, the cars represent the different agents. From any position the agent can choose between 5 actions which correspond to the 4 cardinal directions

and standing still. The mapping ψ also uses these directions to map the xy-coordinate of an agent to a specific road, e.g., `onRoad(north)` signifies the agent is on the road on the north of the intersection. The unary predicate `beforeJunction` denotes if the agent is in front of a junction. The action of the agent is represented by the predicate `go`. The state of the traffic light is represented by `tls0` and `tls1`. If a car is approaching from the right, the predicate `carOnTheRight` is true. The ground truth rules (to be learned constraints) consist of not driving off-road, giving priority to the right on a junction without traffic lights, stopping at a red light and, in case of unidirectional traffic, not driving in the opposite direction. These scenarios are built in the SUMO traffic simulator [17] which comes with internal car control algorithms, representing the expert policy π_E, to generate realistic car trajectories. All possible (valid) trajectories are generated with the same probability and all results are averaged over 20 trials with different random seeds.

3.4.2 Induced Hypotheses

In this subsection we carry out a qualitative evaluation of the induced hypotheses. In the towers of Hanoi environment, only one rule is learned which states that it is invalid that a larger disk `V1` is on a disk `V2`:

```
:- V1 < V2; on(V1,V2).
```

Note that disks are numbered in order of decreasing size. For the towers of Hanoi environment the background knowledge and search space are depicted in Figure 3.1. For reasons of space, the complete background knowledge and search space are omitted for the traffic environments. In the traffic environment with priority to the right the following rules are induced:

```
:- not onRoad(V1) : dir(V1).
:- go(north); not onRoad(south).
:- go(south); not onRoad(north).
:- go(east); onRoad(south).
:- go(east); onRoad(north).
:- go(west).
:- carOnTheRight; not go(zero).
```

The first rule states that it is invalid for an agent to be off-road. This rule makes use of a conditional literal and is equivalent to the conjunction of `not onRoad(V1)` for all directions `V1` (`dir(V1)` evaluates true if `V1` is a variable which represents a direction). The following four rules invalidate being in a state on-road and transitioning to a state off the road. Since all roads in this environment are unidirectional, the action of going west is

invalid as stated by the second last rule. The last rule represents the "priority to the right"-rule, i.e., when there is a car on the right, the only valid action is `go(zero)` which represents being stationary. The rules learned in the traffic light controlled junction environment are displayed below:

```
:- not onRoad(V1) : dir(V1).
:- go(north); onRoad(west).
:- go(north); onRoad(east).
:- go(east); onRoad(south).
:- go(east); onRoad(north).
:- go(south); onRoad(east).
:- go(west); onRoad(south).
:- go(west); onRoad(north).
:- go(south); onRoad(west).
:- beforeJunction(west); go(east); tls1.
:- beforeJunction(south); go(north); tls0.
:- beforeJunction(east); go(west); tls1.
:- beforeJunction(north); go(south); tls0.
```

The first rule invalidates being off-road and the following 8 rules prohibit transitioning to a state off-road, similar rules were induced in the environment with priority to the right. The last four rules state cars have to stop at a red traffic light.

3.4.3 Accuracy of the Learned Hypotheses

In this section, we validate the induced rules of the traffic environment by calculating all answer sets of the ASP program represented by the union of the background knowledge and the induced hypothesis. These answer sets represent the state-action pairs which are considered valid by the induced hypothesis and can hence be used to classify a state-action pair as allowed or prohibited. We gather 10 000 trajectories from the SUMO simulator and validated that all correct actions of cars in any of the possible on-road position were observed at least once in this dataset. Accordingly, the ground truth validity of a state-action pair is determined by the presence in at least one of the 10 000 trajectories. We report three statistics. The false positive rate (FPR) is the fraction of the state-action space that is incorrectly prohibited by the set of induced rules. Accordingly, the false negative rate (FNR) is the fraction that is incorrectly allowed. The accuracy is the ratio between the sum of false positive and false negative state-action pairs divided by the total number of state-action pairs.

We calculate these statistics for the traffic light controlled intersection and the intersection with priority to the right environments for increasing num-

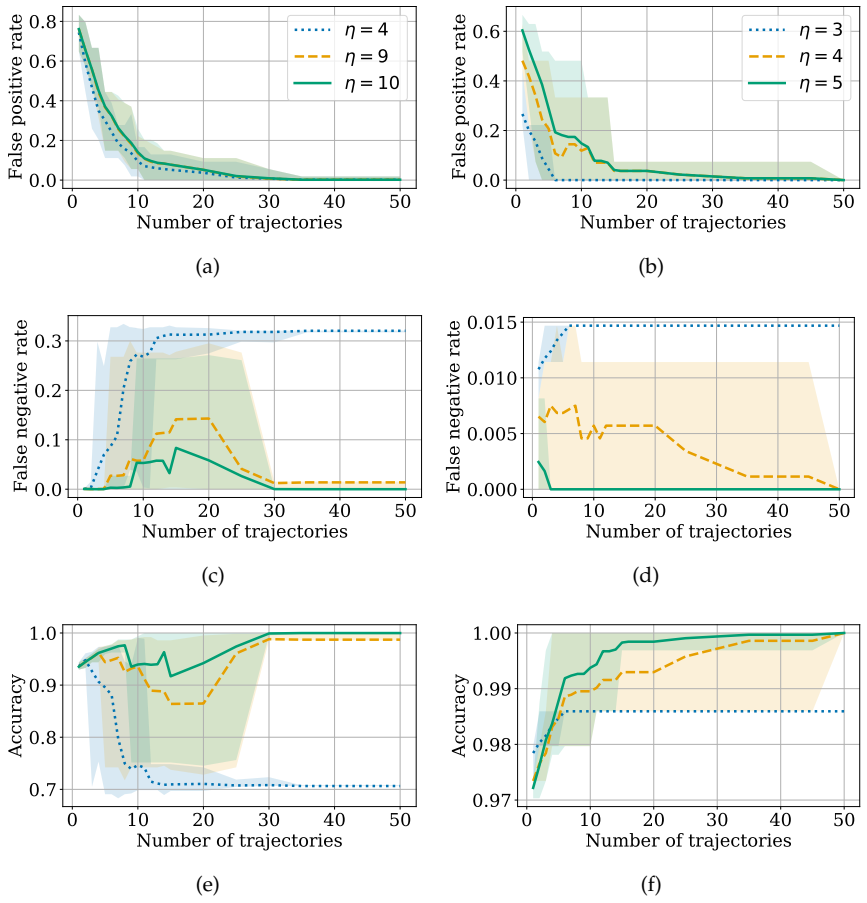


Figure 3.3: False positive rate, false negative rate and accuracy of the induced rules for the traffic light controlled intersection environment (3.3(a), 3.3(c), 3.3(e) respectively) and for the intersection with priority to the right environment (3.3(b), 3.3(d), 3.3(f) respectively) as a function of the number of observed trajectories. Mind the different y-axis scales. The number of iterations is denoted by η.

bers of trajectories (dataset sizes) and iterations η of Algorithm 3.1. The results are depicted in Figure 3.3. The FPR decreases with the number of observed trajectories. When few expert observations are available, some allowed state-action combinations may not be observed. If an allowed but unvisited state-action pair has high likelihood under the nominal policy it will incorrectly be selected as a constraint which will lead to incorrect rules. When, for instance, in none of the trajectories a car stopped in front of a red

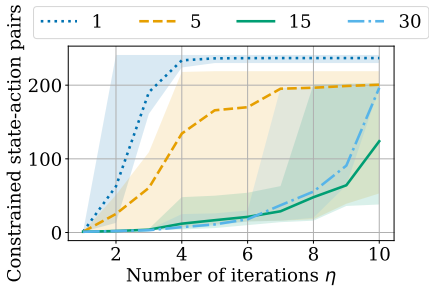


Figure 3.4: Number of state-action pairs which are constrained by the inferred rules for increasing values of η. Line styles indicate the number of expert trajectories available. Results are from the traffic light controlled intersection environment.

traffic light (because the light was always green), the system could induce that it is invalid for a car to be in front of a red traffic light. Provided that the number of iterations η is not too small, small number of observations will result in a low FNR because rules are inferred which invalidate large regions of the state-action space which also include the true constraints. In our example, when the system infers the rule, all states which include the agent in front of a red traffic light are invalid, it also invalidates the true constraint of driving a red light. This phenomenon can also be observed in Figure 3.4 which depicts the number of state-action pairs which are invalidated by the learned rules for increasing values of η and number of expert trajectories. With fewer iterations η, fewer constraints are inferred accordingly the induced rules invalidate less state-action pairs hence a smaller FPR but a higher FNR. It is self-evident that the number of constrained state-action pairs increases when the number of iterations increases. Notably, in the traffic light scenario of Figure 3.3(c), the FNR first increases for smaller number of trajectories. Because the rules become less restrictive, but the number of observations is too limited, many state-action pairs are incorrectly allowed. This is depicted in Figure 3.4 by the green curve (15 trajectories) that lies below the blue dash-dotted curve (30 trajectories). As more observations become available, this effect disappears. This effect is less pronounced for the intersection with priority to the right environment. We attribute this to the lower complexity of this environment with only unidirectional traffic on each road. From these results we conclude there must be sufficient trajectories available and the number of iterations should be high enough to induce a hypothesis with a low FPR, low FNR and high accuracy.

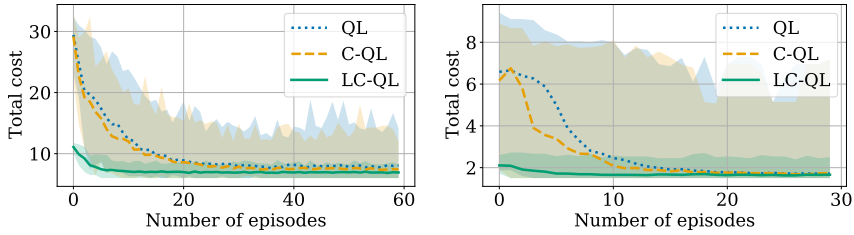


Figure 3.5: Total cost received during training using Q-learning in the traffic light controlled junction environment (3.5(a)) and the intersection with priority to the right environment (3.5(b)).

Table 3.1: Empirical probability of a rule violation during RL experiment.

	Traffic lights		Priority to the right	
	Train	Test	Train	Test
QL	0.32±0.50	0.036±0.072	0.13±0.19	0.0043±0.0140
C-QL	0.25±0.44	0.011±0.039	0.09±0.15	0.0036±0.0150
LC-QL	**0.0±0.0**	**0.0±0.0**	**0.0±0.0**	**0.0±0.0**

3.4.4 Logic-Constrained Q-Learning

We evaluate to what extent the logic program induced by our method can be integrated as prior knowledge in a model-free RL algorithm like Q-learning. Both during training and deployment of RL agents in safety critical environments, we want to ensure the agent integrates in the multi-agent environment and does not cause any harm or damage by violating the social norms and rules. We guide the exploration process by restricting the available actions based on the current state [18, 19]. The action space is restricted by first translating the induced hypothesis to a set of constraints in state-action space (Algorithm 3.1 line 21). Next, the set of sets of valid actions $\{A_s\}$ is restricted using this set of constraints (Algorithm 3.1 line 22). We will refer to this method as logic constrained Q-learning (LC-QL). While this method is conservative, it has the advantage state-action pairs which are forbidden by the induced logic will never occur in any trajectory of the learning agent.

We compare our method with vanilla Q-learning (QL) and Q-learning constrained by a set of constraints obtained using only the inference step of our algorithm (C-QL). Comparing LC-QL and C-QL should reveal the im-

portance of the induction step of Algorithm 3.1. We train both LC-QL and C-QL agents with the same number of iterations η. All agents adopt an ϵ-greedy exploration strategy during training ($\epsilon = 0.05$), but become fully deterministic afterward ($\epsilon = 0$). We adopt a learning rate of 0.9 and a discount factor 0.75. The Q-table is initialized with zeros. Experimental results are the average of 400 trained agents. We define the cost signal (negative reward) as the L1-distance from the agent's position to its goal, which has a maximum of 4. Violating a ground truth rule (i.e., traffic regulations) incurs a cost of 10. Figure 3.5 shows the total cost received by an agent trained for a different number of episodes in the intersection with traffic lights environment (3.5(a)) and the intersection with priority to the right environment (3.5(b)). We conclude from these results that constraining an agent with a logic program inferred using our method leads to a considerable speed-up of the Q-learning convergence. Using only constraint inference barely improves the agent's required number of episodes during training which confirms the importance of the induction step. Moreover, we notice a much smaller variation in performance between LC-QL agents. Table 3.1 shows the empirical probability of an agent violating at least one rule during and after training for both intersections. Best results are in bold. These results show that our method is the only one to guarantee that no rules will be violated both at deployment and during exploration, thus, would qualify for safety-critical applications. At last, we measured the execution time of the different steps of our algorithm, the results for the traffic light controlled junction environment are depicted in Figure 3.6. We observe that the induction step takes up the largest part of the execution time, especially for large values of η. The green curve represents the time until convergence of LC-QL when provided with rules inferred using our algorithm for different values of η. Although our method improves the speed to convergence when constraining a learning agent with the learned rules (as shown in Figures 3.5(a) and 3.5(b)) there is no overall time saving compared to training an agent without prior knowledge.

3.4.5 Transfer Learning

At last, we tested to what extent rules from one environment can be transferred to a second, similar environment. To this end, we designed a traffic situation which combines elements from the traffic light controlled intersection and the intersection with priority to the right environments. A snapshot of this environment is shown in Figure 3.2(c). Since the learned rules are expressed in terms of human concepts which generalize multiple states and actions, it is straightforward to port these rules to another environment. The only manual work required is to update the definition of predicates in

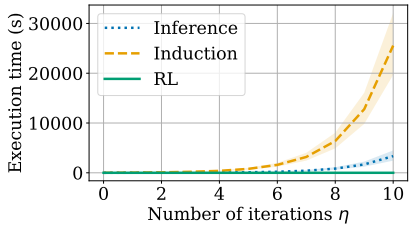


Figure 3.6: Execution time of the different steps of our algorithm applied to the traffic light controlled intersection environment.

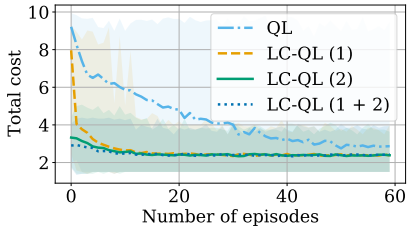


Figure 3.7: Transfer learning experiment: total cost received in the combo environment (see Figure 3.2(c)).

the background knowledge because for example the position of roads and intersections in the source and target domain does not necessarily correspond with the same xy-coordinates (symbol grounding). We compare an agent which learns from scratch (standard Q-learning), an agent which is provided with rules induced from the traffic light controlled intersection environment (LC-QL 1), an agent which is provided with rules induced from the intersection with priority to the right environment (LC-QL 2) and an agent which is provided with the rules induced from both environments (LC-QL 1 + 2). Figure 3.7 depicts the total cost acquired during one run by an agent. From this figure, we can conclude transferring rules induced with our method to a similar environment can substantially reduce the number of episodes required during training, even when the source domain only contains a subset of the rules applicable in the target domain. For example, the agent which only gets rules from source domain 2 (LC-QL (2)) has not learned the rule of stopping at red traffic lights but still trains significantly faster than the QL agent. Table 3.2 shows the empirical probability of one of the agents violating a rule. Best results are in bold. Notably, an agent which is only provided with a portion of the ground truth rules, is able to reduce the empirical probability of a rule violation with one or two orders of magnitude during training and even to zero during testing.

3.5 Related Work

The first definition of constrained MDP's was provided by Altman [20] which augments the MDP with a secondary cost function $c : S \times A \mapsto \mathbb{R}$ and a budget $\alpha \geq 0$. Solving this type of constrained MDP's consists of finding a policy which maximizes the expected total discounted reward $J(\pi)$ such that $J^c(\pi) \leq \alpha$ where $J^c(\pi)$ denotes the expected total discounted cost. However, solving this class of constrained MDP's is not trivial and

Table 3.2: Empirical probability of a rule violation during transfer learning experiment.

	Train	Test
QL	0.220±0.180	0.024±0.025
LC-QL (1)	0.018±0.088	**0.0±0.0**
LC-QL (2)	0.004±0.008	**0.0±0.0**
LC-QL (1 + 2)	**0.0±0.0**	**0.0±0.0**

research is still ongoing [21, 22]. Scobee et al. [7] proposed an alternative definition of constrained MDP's where constraints are defined as invalid state-action pairs, this relaxes the necessity of sophisticated methods to solve the MDP. Hence, the definition of Scobee et al. was evaluated to be best suited for this work.

Several approaches to infer constraints have been proposed. Robotic constraints are often specified as volumes in a 3-dimensional Euclidean space or as kinematic restrictions. Numerous methods were proposed to infer such geometric constraints where demonstrations are provided by teleoperation [23] or by a human guiding the robot [24, 25]. Another study focuses on inferring sequential constraints which capture a task's sequential structure [26]. Previous work [27, 7, 28] defines a constraint as a point in feature space which does not occur during any of the observations but would induce a lower cost. They argue that these points must be constraints, otherwise agents would not avoid these points when minimizing their cost. Our work extends this principle by reasoning about the inferred constraints such that constraints can be generalized over larger regions in feature space. Malik et al. [8] propose a sample based approximation of the work of Scobee et al. by estimating $\mathbb{1}^{\mathcal{M}^C}$ with a neural network. This approach scales constraint inference to large and continuous state spaces. Glazier et al. [29], on the other hand, extend their method to allow soft constraints which are points in feature space which are, in some cases, valid. McPherson et al. [30] adopted the principle of maximum causal entropy to extend the applicability to stochastic environments.

IRL often serves the goal of aligning an artificial agent's behavior with values respected during demonstrations. Noothigattu et al. [31] propose to learn two policies, one maximizing the reward through classic RL and one obeying behavioral constraints through IRL. Next, a contextual-bandit-based orchestrator is used to blend the two policies. Other work introduces an approach to model the navigation behavior of interacting pedestrians in

terms of a joint mixture distribution over the trajectories of all agents [32]. Next, this model is adopted by a robot to interact with humans in a socially compliant way. However, no existing work focuses on true interpretability of the learned concepts which is, by our opinion, a critical aspect in the development of safe artificial agents.

Our work is also related to safe RL as the induced hypothesis can be adopted as external knowledge to avoid unsafe situations during the exploration process. The exploration process can also be modified by initializing the Q-function with recorded trajectories of a teacher [33]. Other work suggests decomposing the Q-function in a task and a survival component [34]. The survival component is task-independent and is used to safely navigate the environment. Other techniques take risks into account by adapting the optimization criteria [35]. However, in contrast with the proposed method, these techniques do not completely exclude the occurrence of an invalid state-action pair. Related to our approach is the technique of a teacher giving advice to the agent when unsafe situations could occur [19]. In our case the teacher is represented by the learned rules which restricts the possible actions in certain states. Safety is also closely related to interpretability which is why other research focuses on explaining RL policies. Coppens et al. [36] proposed a policy distilling algorithm which extracts a set of rules from a deep RL policy. However, in contrast to our work, they assume a RL policy is already available. Alternatively, relational RL [37], learns an optimal policy starting from a description of the environment based on objects and relations which makes it interpretable "by design". It is also possible to learn a reward function [38] or a policy [39] in the relational domain, given expert demonstrations. However, these approaches build on supervised learning techniques which fail when demonstrations are suboptimal. Even though these relational RL models are very expressive, training them is difficult which limits their applicability.

Other ILP techniques have also focused on inducing general rules from observed traces. Inductive general game playing is a technique to induce rules from traces from a wide range of games [40]. However, negative examples are generated from the closed world assumption: all atoms which are not known to be true are assumed to be false. We do not make this assumption since there is a chance that a valid state-action pair is not observed in a finite set of traces. Learning from interpretation transition [41] is a subclass of ILP which learns a transition model from traces of a system. The Apperception Engine [42] builds on this principle but imposes extra conditions of unity on the induced logic program. Our work differentiates from this approach since we are not interested in learning a conclusive

causal theory from the observed traces. Instead, our method induces a compact set of rules to represent the environmental restrictions on the behavior of agents.

3.6 Conclusion and Future Work

We showed that constraint inference and ILP are complementary and provided a framework which integrates them. Thanks to constraint inference, there is no need to collect negative examples while ILP offers the capability to generalize sets of constrained state-action pairs to logic statements hence improve interpretability and transferability. Our method learns a logic program from observed trajectories and represents implicit restrictions on the behavior of the observed agents. These restrictions correspond to the explicit and implicit social norms applicable to the observed environment. We provided a Q-learning agent with the learned rules as prior knowledge about the environment to align its behavior with the applicable social norms and reduce the required number of episodes during training. Moreover, since the learned rules are expressed in formal logic, this facilitates validation by a human to assure no undesired behavior is extracted from the observations. This is an important step towards deploying autonomous agents in environments shared with humans.

Calculating the state-action visitation frequencies during the constraint inference stage requires iterating over all states and actions. This also holds for the backward pass which is used for calculating the nominal policy. In its current inception, the framework is thus not scalable to larger environments. However, instead of exactly calculating the state-action visitation frequencies, we could estimate this by sampling trajectories from the nominal policy and counting state-action visitations. To obtain the optimal nominal policy, a deep model-free RL algorithm like Rainbow [43] or PPO [44] can be used. We hypothesize that these measures will enable scaling the inference step to more complex RL benchmarks. Another requirement is a description of the background knowledge to introduce human concepts (e.g., roads, intersections, traffic lights). To enable scalability of the induction step, it is important to keep the hypothesis space limited. We hypothesize that an egocentric representation of the agent's state and action could facilitate this. Although Q-learning is one of the foundational methods for model-free RL, other state-of-the-art deep RL methods have outperformed Q-learning such as TRPO [45], PPO [44] and Rainbow [43]. Providing logic rules as constraints to a deep RL agent is thus an interesting future directive. The problem of optimizing constrained deep RL methods is often formulated as a dual objective using Lagrangian multipliers [21, 46]: maximize the reward (which represents the goal) and minimize the cost (which represent

the constraints). Representing the cost function by the induced logic program would enable integrating our method with state-of-the-art deep RL algorithms. Another possibility is using a "shield" [19] which monitors the actions taken by the agent and corrects them when the agent would violate a rule. Furthermore, when capturing real-life datasets, it is likely that rule violations will be observed. The use of probabilistic logic to represent weak constraints [29] could relax the assumption no constraint state-action pairs can occur in the observed trajectories. Since we build on the principle of maximum entropy IRL, our method only exactly holds for deterministic environments. Adopting the principle of maximum causal entropy [47] is a potential way to resolve this issue.

References

[1] M. K. Lapinski and R. N. Rimal. *An explication of social norms*. Communication theory, 15(2):127–147, 2005.

[2] B. Christian. *The Alignment Problem: Machine Learning and Human Values*. WW Norton & Company, New York, 2020.

[3] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. *Concrete problems in AI safety*. arXiv preprint arXiv:1606.06565, 2016.

[4] S. Russell. *Human compatible: Artificial intelligence and the problem of control*. Penguin, London, 2019.

[5] T. Everitt and M. Hutter. *Avoiding wireheading with value reinforcement learning*. In International Conference on Artificial General Intelligence, pages 12–22. Springer, 2016.

[6] A. Y. Ng, S. J. Russell, et al. *Algorithms for inverse reinforcement learning.* In Icml, volume 1, page 2, 2000.

[7] D. R. Scobee and S. S. Sastry. *Maximum likelihood constraint inference for inverse reinforcement learning*. arXiv preprint arXiv:1909.05477, 2019.

[8] S. Malik, U. Anwar, A. Aghasi, and A. Ahmed. *Inverse constrained reinforcement learning*. In International Conference on Machine Learning, pages 7390–7399. PMLR, 2021.

[9] S. Muggleton. *Inductive logic programming*. New generation computing, 8(4):295–318, 1991.

[10] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, Cambridge, 2018.

[11] P. Abbeel and A. Y. Ng. *Apprenticeship learning via inverse reinforcement learning*. In Proceedings of the twenty-first international conference on Machine learning, page 1, 2004.

[12] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, et al. *Maximum entropy inverse reinforcement learning*. In Aaai, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[13] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Answer set solving in practice*. Synthesis lectures on artificial intelligence and machine learning, 6(3):1–238, 2012.

[14] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Maratea, F. Ricca, and T. Schaub. *ASP-Core-2 input language format*. Theory and Practice of Logic Programming, 20(2):294–309, 2020.

[15] M. Law, A. Russo, and K. Broda. *The ilasp system for inductive learning of answer set programs*. arXiv preprint arXiv:2005.00904, 2020.

[16] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. *Multi-shot ASP solving with clingo*. CoRR, abs/1705.09811, 2017.

[17] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner. *Microscopic Traffic Simulation using SUMO*. In The 21st IEEE International Conference on Intelligent Transportation Systems. IEEE, 2018.

[18] J. Garcıa and F. Fernández. *A comprehensive survey on safe reinforcement learning*. Journal of Machine Learning Research, 16(1):1437–1480, 2015.

[19] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. *Safe reinforcement learning via shielding*. In Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[20] E. Altman. *Constrained Markov decision processes: stochastic modeling*. Routledge, London, 1999.

[21] C. Tessler, D. J. Mankowitz, and S. Mannor. *Reward constrained policy optimization*. arXiv preprint arXiv:1805.11074, 2018.

[22] A. Wachi and Y. Sui. *Safe reinforcement learning in constrained markov decision processes*. In International Conference on Machine Learning, pages 9797–9806. PMLR, 2020.

[23] C. Pérez-D'Arpino and J. A. Shah. *C-learn: Learning geometric constraints from demonstrations for multi-step manipulation in shared autonomy*. In 2017 IEEE International Conference on Robotics and Automation (ICRA), pages 4058–4065. IEEE, 2017.

[24] L. Armesto, J. Bosga, V. Ivan, and S. Vijayakumar. *Efficient learning of constraints and generic null space policies*. In 2017 IEEE International Conference on Robotics and Automation (ICRA), pages 1520–1526. IEEE, 2017.

[25] G. Subramani, M. Zinn, and M. Gleicher. *Inferring geometric constraints in human demonstrations*. In Conference on Robot Learning, pages 223–236. PMLR, 2018.

[26] M. Pardowitz, R. Zöllner, and R. Dillmann. *Learning sequential constraints of tasks from user demonstrations.* In Humanoids, pages 424–429, 2005.

[27] G. Chou, D. Berenson, and N. Ozay. *Learning constraints from demonstrations*. arXiv preprint arXiv:1812.07084, 2018.

[28] G. Chou, N. Ozay, and D. Berenson. *Learning constraints from locally-optimal demonstrations under cost function uncertainty*. IEEE Robotics and Automation Letters, 5(2):3682–3690, 2020.

[29] A. Glazier, A. Loreggia, N. Mattei, T. Rahgooy, F. Rossi, and K. B. Venable. *Making Human-Like Trade-offs in Constrained Environments by Learning from Demonstrations*. arXiv preprint arXiv:2109.11018, 2021.

[30] D. L. McPherson, K. C. Stocking, and S. S. Sastry. *Maximum Likelihood Constraint Inference from Stochastic Demonstrations*. In 2021 IEEE Conference on Control Technology and Applications (CCTA), pages 1208–1213. IEEE, 2021.

[31] R. Noothigattu, D. Bouneffouf, N. Mattei, R. Chandra, P. Madan, K. Varshney, M. Campbell, M. Singh, and F. Rossi. *Interpretable multi-objective reinforcement learning through policy orchestration*. arXiv preprint arXiv:1809.08343, 2018.

[32] H. Kretzschmar, M. Spies, C. Sprunk, and W. Burgard. *Socially compliant mobile robot navigation via inverse reinforcement learning*. The International Journal of Robotics Research, 35(11):1289–1307, 2016.

[33] J. de Lope et al. *Learning autonomous helicopter flight with evolutionary reinforcement learning*. In International Conference on Computer Aided Systems Theory, pages 75–82. Springer, 2009.

[34] P. Van Molle, T. Verbelen, S. Bohez, S. Leroux, P. Simoens, and B. Dhoedt. *Decoupled Learning of Environment Characteristics for Safe Exploration*. arXiv preprint arXiv:1708.02838, 2017.

[35] P. Geibel. *Reinforcement learning for MDPs with constraints*. In European Conference on Machine Learning, pages 646–653. Springer, 2006.

[36] Y. Coppens, D. Steckelmacher, C. M. Jonker, and A. Nowé. *Synthesising Reinforcement Learning Policies Through Set-Valued Inductive Rule Learning*. In TAILOR, pages 163–179, 2020.

[37] S. Džeroski, L. De Raedt, and K. Driessens. *Relational reinforcement learning*. Machine learning, 43(1):7–52, 2001.

[38] T. Munzer, B. Piot, M. Geist, O. Pietquin, and M. Lopes. *Inverse reinforcement learning in relational domains*. In Twenty-Fourth International Joint Conference on Artificial Intelligence, 2015.

[39] S. Natarajan, S. Joshi, P. Tadepalli, K. Kersting, and J. Shavlik. *Imitation learning in relational domains: A functional-gradient boosting approach*. In Twenty-Second International Joint Conference on Artificial Intelligence, 2011.

[40] A. Cropper, R. Evans, and M. Law. *Inductive general game playing*. Machine Learning, 109(7):1393–1434, 2020.

[41] K. Inoue, T. Ribeiro, and C. Sakama. *Learning from interpretation transition*. Machine Learning, 94(1):51–79, 2014.

[42] R. Evans, J. Hernández-Orallo, J. Welbl, P. Kohli, and M. Sergot. *Making sense of sensory input*. Artificial Intelligence, 293:103438, 2021.

[43] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. *Rainbow: Combining improvements in deep reinforcement learning*. In Thirty-second AAAI conference on artificial intelligence, 2018.

[44] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. *Proximal policy optimization algorithms*. arXiv preprint arXiv:1707.06347, 2017.

[45] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. *Trust region policy optimization*. In International conference on machine learning, pages 1889–1897. PMLR, 2015.

[46] G. Kalweit, M. Huegle, M. Werling, and J. Boedecker. *Deep constrained q-learning*. arXiv preprint arXiv:2003.09398, 2020.

[47] B. D. Ziebart, J. A. Bagnell, and A. K. Dey. *Modeling interaction via the principle of maximum causal entropy*. In ICML, 2010.

3.7 Appendix

3.7.1 Qualitative Results

This section provides the background knowledge B, the language bias M defining the search space S_M and the induced hypotheses H for the different environments used in the experiments.

3.7.1.1 Towers of Hanoi

Background B

```
y(0..2).
x(0..2).
disk(0..2).

1{at(D, Y, X) : y(Y), x(X)}1 :- disk(D).
on(D1, D2) :- at(D1, X1, Y1), D1!=D2, at(D2, X2, Y2), X1=X2, Y1>Y2.
```

The first three lines define constants representing the x- and y-coordinates and the disks. The rule on line 5 states, only one disk can be at a single position defined with an x- and y-coordinate. The last rule defines the predicate `on(D1,D2)` which evaluates to true when disk `D1` is on disk `D2`.

Language bias M

```
#modeb(1, on(var(disk), var(disk))).
#modeb(1, var(disk)=var(disk)).
#modeb(1, var(disk)<var(disk)).
#modeb(1, var(disk)>var(disk)).
```

Hypothesis H

```
:- V1 < V2; on(V1,V2).
```

3.7.1.2 Traffic Light Controlled Intersection

Background B

```
row(1..5).
col(1..5).

dir(zero).
dir(north).
dir(east).
dir(south).
dir(west).
```

```
1{at(C, R): col(C), row(R)}1.
1{tls0; tls1}1.
1{go(V) : dir(V)}1.

onRoad(zero) :- at(X,Y), col(X), row(Y), X=3, Y=3.
onRoad(south) :- at(X,Y), col(X), row(Y), X=3, Y<3.
onRoad(north) :- at(X,Y), col(X), row(Y), X=3, Y>3.
onRoad(east) :- at(X,Y), col(X), row(Y), X>3, Y=3.
onRoad(west) :- at(X,Y), col(X), row(Y), X<3, Y=3.

beforeJunction(south) :- at(X,Y), col(X), row(Y), X=3, Y=2.
beforeJunction(north) :- at(X,Y), col(X), row(Y), X=3, Y=4.
beforeJunction(west) :- at(X,Y), col(X), row(Y), X=2, Y=3.
beforeJunction(east) :- at(X,Y), col(X), row(Y), X=4, Y=3.
```

Line 1 and 2 define the x- and y-coordinates. Line 4 to 8 define the directions used to indicate actions and roads. The rule on line 10 states only one agent can be at a single position. The rule on line 11 defines that the traffic light is always in one of the two states: `tls0` and `tls1`. The rule on line 12 assures an agent can only move in one direction. The other rules are explained in figure 3.8.

Language bias M

```
#constant(direction, zero).
#constant(direction, north).
#constant(direction, east).
#constant(direction, south).
#constant(direction, west).

#constant(cord, 1).
#constant(cord, 2).
#constant(cord, 3).
#constant(cord, 4).
#constant(cord, 5).

#modeb(1, onRoad(const(direction))).
#modeb(1, tls0, (positive)).
#modeb(1, tls1, (positive)).
#modeb(1, go(const(direction))).
#modeb(1, beforeJunction(const(direction))).
#modeb(1, at(const(cord), const(cord))).
#modeb(1, onRoad(var(dir))).
```

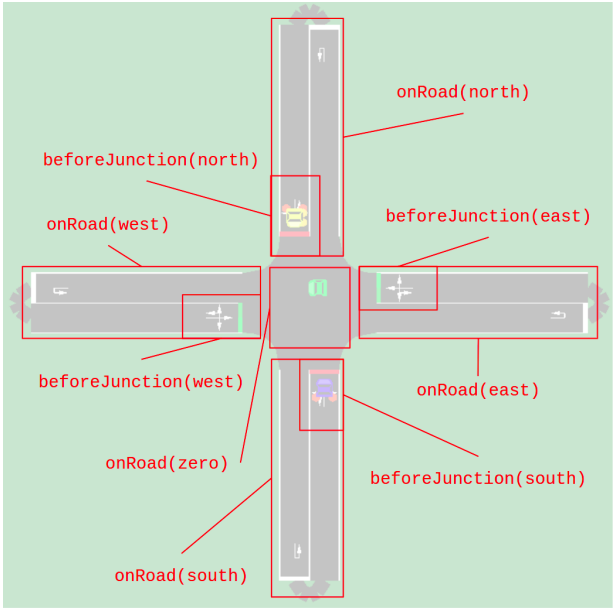


Figure 3.8: Meaning of predicates in the traffic light controlled intersection environment.

```
#modec(1, dir(var(dir))).

#max_penalty(50).
```

Hypothesis H

```
:- not onRoad(V1) : dir(V1).
:- go(north); onRoad(west).
:- go(north); onRoad(east).
:- go(east); onRoad(south).
:- go(east); onRoad(north).
:- go(south); onRoad(east).
:- go(west); onRoad(south).
:- go(west); onRoad(north).
:- go(south); onRoad(west).
:- beforeJunction(west); go(east); tls1.
:- beforeJunction(south); go(north); tls0.
:- beforeJunction(east); go(west); tls1.
:- beforeJunction(north); go(south); tls0.
```

3.7.1.3 Intersection with Priority to the Right

Background B

```
row(1..5).
col(1..5).

1{at(C, R): col(C), row(R)}1.
1{carOnTheRight; noCarOnTheRight}1.
1{go(V) : dir(V)}1.

dir(zero).
dir(north).
dir(east).
dir(south).
dir(west).

onRoad(zero) :- at(X,Y), col(X), row(Y), X=3, Y=3.
onRoad(south) :- at(X,Y), col(X), row(Y), X=3, Y<3.
onRoad(north) :- at(X,Y), col(X), row(Y), X=3, Y>3.
onRoad(east) :- at(X,Y), col(X), row(Y), X>3, Y=3.
onRoad(west) :- at(X,Y), col(X), row(Y), X<3, Y=3.
```

The background knowledge is similar to the traffic light controlled intersection. In this environment, the predicates `carOnTheRight` and `NoCarOnTheRight` are introduced.

Language bias M

```
#constant(direction, zero).
#constant(direction, north).
#constant(direction, east).
#constant(direction, south).
#constant(direction, west).

#constant(cord, 1).
#constant(cord, 2).
#constant(cord, 3).
#constant(cord, 4).
#constant(cord, 5).


#modeb(1, at(const(cord), const(cord)), (positive)).
#modeb(1, onRoad(const(direction))).
```

```
#modeb(1, go(const(direction))).
#modeb(1, carOnTheRight).
#modeb(1, onRoad(var(dir))).
#modec(1, dir(var(dir))).

#max_penalty(30).
```

Hypothesis H

```
:- not onRoad(V1) : dir(V1).
:- go(north); not onRoad(south).
:- go(south); not onRoad(north).
:- go(east); onRoad(south).
:- go(east); onRoad(north).
:- go(west).
:- carOnTheRight; not go(zero).
```

3.7.2 Transfer Learning

As mentioned in the article, the meaning (grounding) of predicates can differ between the source and target domain. Below the updated background knowledge is given used in the transfer learning experiment.

Background B

```
row(1..7).
col(1..7).

1{at(C, R): col(C), row(R)}1.
1{tls0; tls1}1.
1{carOnTheRight; noCarOnTheRight}1.
1{go(V) : dir(V)}1.

dir(zero).
dir(north).
dir(east).
dir(south).
dir(west).

onRoad(zero) :- at(X,Y), col(X), row(Y), X=6, Y=4.
onRoad(south) :- at(X,Y), col(X), row(Y), X=3, Y<4.
onRoad(north) :- at(X,Y), col(X), row(Y), X=6, Y>4.
onRoad(east) :- at(X,Y), col(X), row(Y), X>6, Y=4.
onRoad(west) :- at(X,Y), col(X), row(Y), X<6, Y=4.
```

```
beforeJunction(north) :- at(X,Y), col(X), row(Y), X=6, Y=5.
beforeJunction(west) :- at(X,Y), col(X), row(Y), X=5, Y=4.
beforeJunction(east) :- at(X,Y), col(X), row(Y), X=7, Y=7.
beforeJunction(south) :- at(X,Y), col(X), row(Y), X=7, Y=7.
```

4

Constraint Inference Using One-Class Decision Trees

"I know the pieces fit 'cause I watched them fall away."

TOOL, "Schism"

Learning Safety Constraints From Demonstration Using One-Class Decision Trees

M. Baert • S. Leroux • P. Simoens.
Presented at the AAAI workshop on Neuro-Symbolic Learning and Reasoning in the Era of Large Language Models (NucLeaR), 2024.

In the previous chapter, we introduced an ICRL method that leverages Inductive Logic Programming (ILP) to learn constraints as first-order logic formulas. While this approach offers high interpretability, it is limited to relatively simple environments. To extend symbolic constraint learning to more complex settings, this chapter reframes the ICRL problem as a one-class classification task. We leverage one-class decision trees to accurately capture the state distribution of expert demonstrations. These decision trees provide a foundation for representing a set of constraints pertinent to the given environment as a logical formula in disjunctive normal form. The learned constraints are subsequently used within a constrained reinforcement learning framework, enabling the acquisition of a safe policy. In contrast to other methods, our approach offers an interpretable representation of the constraints, a vital feature in safety-critical environments. To validate the effectiveness of our proposed method, we conduct experiments in synthetic benchmark domains and a realistic driving environment.

4.1 Introduction

Reinforcement Learning (RL) has made significant strides in training autonomous agents, but as these systems become more advanced, ensuring their safety and alignment with human intentions, often referred to as the alignment problem [1], is becoming a critical concern. As a result, the objectives of autonomous agents extend beyond their primary task goals, also encompassing safety constraints, human values, legal regulations, and various other factors. For instance, an autonomous vehicle's objective is not only to get passengers quickly to their destination but also to abide by traffic laws and ensure passenger comfort. To address these challenges and enhance the safety of RL agents, Constrained Reinforcement Learning (CRL) methods have emerged. CRL is designed to obtain safe RL agents by incorporating constraints into the learning process. CRL methods help ensure that the RL agent operates within predefined boundaries, reducing the risk of harmful or unintended actions. However, one key limitation of CRL methods is that they assume the availability of a well-defined set of constraints. Specifying these constraints correctly and completely is a complex task on its own [2]. It often requires a deep understanding of the environment, potential risks, and ethical considerations. Therefore, addressing the alignment problem in RL

goes beyond the development of CRL methods; it also involves addressing the challenge of correctly specifying constraints as necessary. This chapter introduces a novel approach to reduce the need for manual specification of constraints by learning from expert demonstrations. First, a model of the expert behavior and its underlying constraints is learned as a one-class decision tree [3]. From this tree a logical formula in disjunctive normal form is extracted defining the constraints. Next, we demonstrate that the learned constraints can be used by a CRL method to learn a constraint-abiding (i.e., safe) policy in synthetic and realistic environments. A notable advantage of this approach is the interpretability of both the learned expert model and the constraints themselves. By monitoring the evaluation-violation ratio of the constraints during training, we can refine the set of constraints after training, further enhancing interpretability. Additionally, as constraints are frequently shared across multiple agents and tasks, once they have been acquired, they can be seamlessly applied to other agents undertaking diverse tasks. This eliminates the need to separately learn constraints for each individual agent and task. We assess the effectiveness of the proposed approach through an evaluation on a series of synthetic benchmarks for safe RL [4] and in a real-world driving scenario [5].

4.2 Background

4.2.1 Markov Decision Process

A Markov decision process (MDP) is characterized by a state space $\mathcal{S}$, an action space $\mathcal{A}$, a discount factor γ within the range [0,1], a transition distribution denoted as $p(s' \mid s, a)$, an initial state distribution represented by $\mathcal{I}(s)$, and a reward function $R : \mathcal{S} \times \mathcal{A} \mapsto [r_{\min}, r_{\max}]$. Within this framework, an agent interacts with the environment at discrete time steps, generating a sequence of state-action pairs known as a trajectory $\tau = ((s_0, a_0), ..., (s_{T-1}, a_{T-1}))$, where T is the length of the trajectory. The cumulative reward of a trajectory is calculated as the sum of rewards, each discounted by a factor of γ^t, where t represents the time step: $R(\tau) = \sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)$. At each discrete time step, the agent's choice of action is governed by a policy π, which is a function that maps states from the state space $\mathcal{S}$ to a probability distribution over actions from the action space $\mathcal{A}$. The primary objective of forward reinforcement learning is to find a policy π that maximizes the expected sum of discounted rewards, expressed as $J_r(\pi) = \mathbb{E}_{\tau\sim\pi} R(\tau)$.

4.2.2 Constrained Reinforcement Learning

Constraints provide a natural and widely applicable means of specifying safety requirements in various contexts [4]. Within the domain of Con-

strained Reinforcement Learning (CRL), the prevailing framework for problem modeling is the Constrained Markov Decision Process (CMDP) [6]. CMDPs extend the traditional Markov Decision Processes (MDPs) by introducing a non-negative bounded cost function, denoted as $C : \mathcal{S} \times \mathcal{A} \mapsto [c_{\min}, c_{\max}]$, and a budget parameter $\alpha \geq 0$. In this context, $C(s, a)$ quantifies the cost associated with taking action a in state s, while the cumulative cost of a trajectory is defined as the summation of discounted costs: $C(\tau) = \sum_{t=0}^{T-1} \gamma^t C(s_t, a_t)$. The objective in CRL is to find an optimal policy that maximizes the expected sum of discounted rewards, as denoted by $J_r(\pi)$, while adhering to specific constraints defined by the cost function C. Formally, the optimal policy π^* is obtained through the following optimization problem:

$$\pi^* = \arg\max_{\pi} J_r(\pi) \text{ s.t. } J_c(\pi) < \alpha. \tag{4.1}$$

Here, $J_c(\pi) = \mathbb{E}_{\tau \sim \pi} C(\tau)$ represents the expected sum of discounted costs, and α signifies a limit on the allowable cost.

4.2.3 One-Class Decision Tree

One-class classification (OCC) is a machine learning paradigm that involves training a model to classify instances into a single well-defined class, treating all other data points as anomalies or outliers. One-class trees (OC-trees) [3] provide an interpretable approach for addressing OCC problems. Building on kernel density estimation, OC-trees aim to represent target areas in the input space that describe the training data. We consider the training data as a set of M instances $X = \{x_0, x_1, \ldots x_{M-1}\}$. Each instance is characterized by a k-dimensional feature vector, with x_i^j representing the j-th feature of the i-th instance. We define $\chi \subset \mathbb{R}^k$ as a k-dimensional hyper-rectangle that encompasses all training instances. The primary objective is to partition the initial hyper-rectangle χ into distinct subspaces χ_t, represented by tree nodes t, such that the learned sub-spaces encompass the training data. The subspace χ_t associated with node t is divided into one or more sub-spaces $\chi_{t_n} = \{x \in \chi_t : \mathrm{L}_{t_n} \leq x^j \leq \mathrm{R}_{t_n}\}$ with $L_{t_n}, R_{t_n} \in \mathbb{R}$, $n \in \{0, ..., N-1\}$ and N the number of children of t. At a given node t, several steps are carried out for each dimension to determine the dimension j which best cuts the data in multiple sub-spaces:

- Estimate the probability density function $\hat{f}_j(x)$ along dimension j using kernel density estimation, based on the available training samples $x \in \chi_t$.
- Divide the subspace χ_t along dimension j based on the modes of the estimated density function $\hat{f}_j(x)$ into N intervals defined by their left

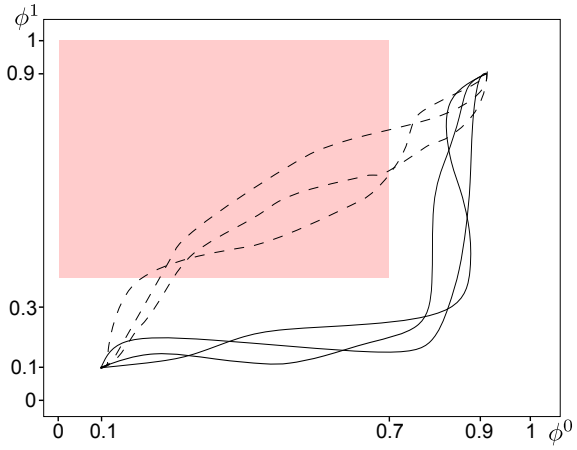


Figure 4.1: Simple navigation environment with a two-dimensional feature space. The red region represents the ground truth constraints. The solid lines represent expert trajectories and the dashed line, trajectories a learning agent would take which is unaware of the constraints.

and right bound L_{t_n} and R_{t_n}. Note that N depends on the number of modes of $\hat{f}_j(x)$.

- Evaluate the quality of the division using a measure of *impurity*.

The dimension that yields the best purity score is selected to partition the subspace χ_t.

4.3 Method

Our approach builds upon the insight, as discussed by Lindner et al. [7], that feature expectations of policies adhering to the genuine constraints form a safe set within the feature vector space. If, for some policy, we can guarantee that the feature expectations are enclosed by the safe set then we can guarantee this policy is safe. Our method encompasses four key steps: first, a safe set is established through constructing an OC-tree; following this, constraints are extracted in the form of a logical formula; then, we use a CRL method to train a policy that conforms to these constraints; and finally, we propose an approach for pruning the learned formula after training.

4.3.1 Learning a Safe Set

In this section, we discuss the process of acquiring a representation of safe behavior. We start with a collection of expert trajectories denoted as $\mathcal{T}$. We define a fixed mapping from state-action tuples to a bounded k-dimensional feature space as $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^k$. Our goal is to learn a safe set in this

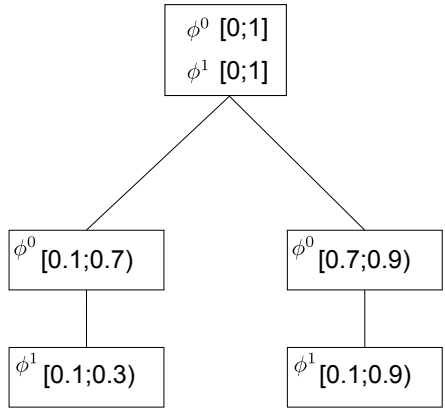


Figure 4.2: Expert behavior tree learned from trajectories presented in figure 4.1. The root of tree encompasses the complete feature space. Every other node defines an interval along one dimension such that the complete tree represents the safe set as multiple hyper-rectangles. This tree represents two rectangles in the two-dimensional feature space.

feature space. To accomplish this, we construct a dataset $\mathcal{D}$ from the set of trajectories, defined as: $\mathcal{D} = \{\phi(s,a) : \forall\ (s,a) \in \tau; \forall\ \tau \in \mathcal{T}\}$. Training an OC-tree [3] on the dataset $\mathcal{D}$ constructs a region in the feature space by taking the union of hyper-rectangles. To illustrate this concept, let's consider a simplified environment characterized by a two-dimensional feature space. In this scenario, the first dimension, denoted as ϕ^0, corresponds to the agent's x-coordinate, while the second dimension, ϕ^1, represents the agent's y-coordinate. Figure 4.1 visually represents this environment along with three expert trajectories depicted as solid lines. The agent is tasked with a straightforward navigation assignment, starting from an initial feature vector of $[0.1, 0.1]$ and aiming to reach a goal vector of $[0.9, 0.9]$. The red rectangle in Figure 4.1 represents a constrained area within the feature space. Figure 4.2 showcases the OC-tree that has been learned from the provided trajectories, which we refer to as the expert tree or expert model. In this specific instance, the expert behavior is enclosed by two rectangles within the feature space.

4.3.2 Formula Extraction

Our objective is to acquire a logic formula φ in Disjunctive Normal Form (DNF) that captures the constraints governing the expert's behavior. φ should evaluate true for the portion of the feature space not covered by the safe set represented by the expert tree. For a tree defined by its root node t and a given feature vector ϕ, the corresponding formula φ_t is recursively

defined as follows:

$$\begin{aligned}\varphi_t = &\bot \vee \bigvee_{t_{\text{child}} \in t} \Big((\phi^j < L_{t_{\text{child}}}) \vee (\phi^j > R_{t_{\text{child}}}) \\ &\vee \big((\phi^j \geq L_{t_{\text{child}}}) \wedge (\phi^j \leq R_{t_{\text{child}}}) \wedge \varphi_{t_{\text{child}}} \big) \Big).\end{aligned} \tag{4.2}$$

In this context, j denotes the split dimension corresponding to node t. Note that φ_t evaluates to $\bot$ if t is a leaf node. When φ evaluates to $\top$, it signifies that the input feature vector corresponds to a state-action tuple that violates the constraints. For the simple navigation examples the following formula can be extracted from the expert tree presented in Figure 4.2:

$$\begin{aligned}\varphi = &(\phi^0 < 0.1) \vee (\phi^0 > 0.9) \\ &\vee \Big((\phi^0 > 0.1) \wedge (\phi^0 < 0.7) \wedge (\phi^1 < 0.1) \Big) \\ &\vee \Big((\phi^0 > 0.1) \wedge (\phi^0 < 0.7) \wedge (\phi^1 > 0.3) \Big) \\ &\vee \Big((\phi^0 > 0.7) \wedge (\phi^0 < 0.9) \wedge (\phi^1 < 0.1) \Big) \\ &\vee \Big((\phi^0 > 0.7) \wedge (\phi^0 < 0.9) \wedge (\phi^1 > 0.9) \Big).\end{aligned}$$

Additionally, for each dimension j, two more rules are incorporated to invalidate feature representations where ϕ^j is lower than the observed minimum in the expert trajectories: $\min(\phi^j : \forall \phi \in \mathcal{D})$ or higher than the observed maximum: $\max(\phi^j : \forall \phi \in \mathcal{D})$.

4.3.3 Constraint-Based Cost Function and Optimization

The extracted formula φ gives rise to a cost function that can be used within a CRL framework. In cases where a constraint is violated, the agent incurs a cost of 1; conversely, if all constraints are adhered to, the cost remains at 0. To tackle this constrained problem, we employ the Lagrangian method in conjunction with Proximal Policy Optimization (PPO) [8]. This constrained problem can then be addressed as an unconstrained max-min optimization problem, and we adopt the implementation provided by the Omnisafe framework [9]. It's noteworthy that the PPO-Lagrangian approach, while conservative, has been shown to yield comparable or superior results to other methods such as Constrained Policy Optimization (CPO) [10], Projected Constrained Policy Optimization (PCPO) [11], and First-Order Constrained Policy Optimization with Penalty (FOCOPS) [12] on various safety gym benchmark environments, as discussed by Ji et al. [9] and Ray et al. [4].

4.3.4 Refining Constraint Definitions

It is important to mention that the learned tree could serve directly as a cost function. However, if our intention is to prune the set of constraints to improve interpretability, the DNF description of the constraints becomes more practical. We interpret each conjunction in φ as an individual rule or constraint. During the training of the CRL agent, we maintain a record of the number of times each conjunction is evaluated and how many times it evaluates to true (i.e., the corresponding constraint is violated). Conjunctions that exhibit a violation-evaluation ratio below a predefined threshold are pruned from the constraint definition. The rationale behind this approach lies in the fact that during training, the learning agent strives to maximize the discounted sum of rewards. When rules are violated during the execution of highly rewarding behavior, it suggests that these rules likely correspond to genuine constraints since the expert agent actively avoided these states. Conversely, when rules are never violated, they are less likely to be true constraints, as they pertain to regions in the feature space that are scarcely visited by the learning agent, indicating low reward potential. When evaluating a feature representation, the conjunctions are assessed in ascending order of complexity. As a result, conjunctions with fewer literals are given preference over more intricate rules. For the simple navigation example, the pruned constraints are defined as follows:

$$\varphi = (\phi^0 > 0.1) \wedge (\phi^0 < 0.7) \wedge (\phi^1 > 0.3).$$

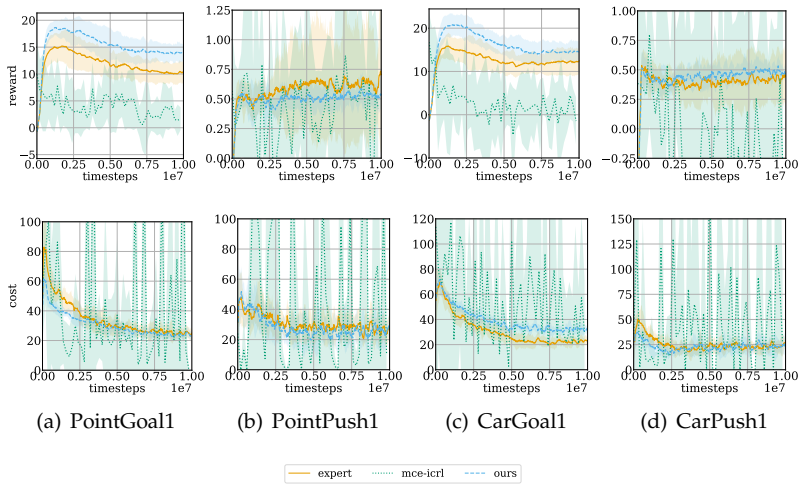
It is worth highlighting that if we examine the trajectories taken by an initial learning agent, as depicted by the dashed lines in Figure 4.1, we will observe a notably high violation-evaluation ratio for this rule. On the contrary the rules which are pruned describe areas within the feature space where the potential rewards are limited, and consequently, these regions are seldom explored by the learning agent.

4.4 Results

We evaluate our method on a set of synthetic safe RL benchmark domains proposed by Ray et al. [4] and a realistic highway environment [5]. All results are averaged over 10 trials with random seeds. In the appendix, we provide examples of the learned formulas.

4.4.1 Synthetic Environments

We assess two categories of agents: The *Point* agent, a basic robot confined to a 2D plane, equipped with two actuators for rotation and forward/backward movement, and the *Car* agent, a somewhat more complex robot with two independently driven parallel wheels and a free-rolling rear



(a) PointGoal1 (b) PointPush1 (c) CarGoal1 (d) CarPush1

Figure 4.3: Reward (top) and ground truth cost (bottom) during training of agents in the synthetic benchmark environments.

wheel. In the case of the *Car* robot, both steering and forward/backward movement necessitate coordination between the two drives. Our evaluation encompasses two distinct tasks: *Goal* and *Push*. In the *Goal* task, the agent's aim is to reach a specified goal location while circumventing hazards. The *Push* task involves the agent pushing a box to the designated goal position while avoiding hazards. The feature space for the various agents is characterized by the agent's acceleration and velocity along both the x and y axes, in addition to the data from 16 LiDAR sensors: $a_x, a_y, v_x, v_y, d_{l0:15}$. The LiDAR sensor readings indicate the distance in each of the 16 directions around the agent to a hazard. The LiDAR values range from 0 to 1, with a reading of 1 signifying that the agent is in a hazardous area.

Figure 4.3 illustrates the acquired rewards and costs during the training of different agents. These agents include one provided with ground truth constraints and trained using PPO-Lagrangian (i.e., expert), another trained with expert demonstrations using Maximum Causal Entropy Inverse Constrained Reinforcement Learning (MCE-ICRL) [13] as a state-of-the-art inverse constrained reinforcement learning method, and an agent trained with expert demonstrations using our proposed method (i.e., ours). We leverage the expert agent's policy, trained using the ground truth constraints, to sample trajectories, which serve as input expert trajectories for both MCE-ICRL and our method. For both the *Goal* and *Push* tasks our method approximates the reward and cost achieved by the expert. The *Push* task

Table 4.1: Transferring constraints between agents and tasks: reward (↑ is better)

Source/target	PointGoal1	PointPush1	CarGoal1	CarPush1
PointGoal1	13.9 ± 1.7	0.6 ± 0.3	4.9 ± 2.1	0.7 ± 0.3
PointPush1	0.5 ± 0.2	0.6 ± 0.1	0.1 ± 0.2	0.4 ± 0.1
CarGoal1	12.2 ± 3.6	0.9 ± 0.3	15.1 ± 3.1	0.7 ± 0.2
CarPush1	0.1 ± 0.1	0.5 ± 0.1	0.3 ± 0.3	0.5 ± 0.1

Table 4.2: Transferring constraints between agents and tasks: ground truth cost (↓ is better)

source/target	PointGoal1	PointPush1	CarGoal1	CarPush1
PointGoal1	27.9 ± 5.0	54.5 ± 16.7	35.1 ± 11.6	47.4 ± 9.3
PointPush1	4.5 ± 1.7	26.2 ± 7.3	6.5 ± 3.9	18.7 ± 8.2
CarGoal1	41.0 ± 8.0	50.3 ± 12.8	33.7 ± 5.3	48.7 ± 9.9
CarPush1	13.9 ± 4.4	25.7 ± 8.3	14.4 ± 3.5	24.6 ± 10.6

presents a significantly greater challenge compared to the *Goal* task, as is reflected by the lower rewards obtained. Even when provided with ground truth constraints, current CRL methods struggle to learn a satisfactory policy. Consequently, the expert trajectories will be suboptimal, inevitably impacting the performance of our method, as these trajectories serve as its fundamental input. In essence, the efficacy of our method is upper bounded by the quality of the expert demonstrations. Furthermore, the same oracle CRL method is part of our proposed approach, which raises concerns that its failure in cases where ground truth constraints are available may also extend to situations where it operates with learned constraints. It is notable that MCE-ICRL fails to recover a satisfactory policy in our experiments. This can be attributed to the complexity of the environments which significantly surpasses those used in the original paper [13].

One key advantage of defining or learning constraints is that they are often the same for various agents and tasks. This is beneficial because we can learn the constraints once and use them in multiple settings. To this end, we assess to what extent the constraints learned from demonstrations of an agent of type A performing task X can be transferred to agents of type B performing task Y. In Table 4.1 and 4.2, we present the reward and cost, respectively, acquired by agents after training. The rows indicate the environment in which the rules are learned (source domain) and the columns correspond with the environment in which the agent is trained (target domain). The main diagonal contains the results when the source

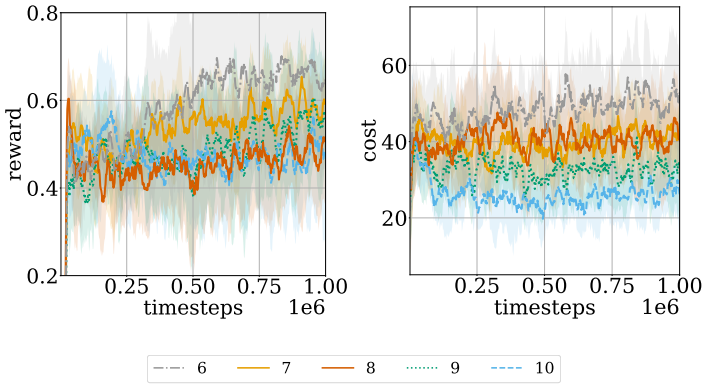


Figure 4.4: Reward (left) and cost (right) during training in the *CarPush* environment for constraints extracted from trees with various depths.

and target domain are the same. The best results for a given target domain are achieved when the source and target domains are the same. Notably, the most favorable outcomes are observed when transferring constraints between different agents performing the same task (e.g. *CarGoal* to *PointGoal*). However, when constraints are transferred between distinct tasks for the same agent, there is a noticeable decline in performance (e.g. *PointPush* to *PointGoal*). Analyzing the learned formula from expert trajectories performing the *Push* task, we find that more restrictive constraints are learned compared to when expert trajectories optimize the *Goal* task. This discrepancy is reflected in the results, where employing constraints learned in the *Goal* domain within the *Push* domain leads to a higher frequency of constraint violations, suggesting that the constraints are excessively lenient. Conversely, employing constraints acquired in the *Push* domain within the *Goal* domain results in lower costs but also reduced rewards, indicating that the constraints are overly restrictive.

For each environment, we determine the ideal tree depth for achieving optimal results. It is important to note that as the tree depth increases, the learned formula becomes more restrictive. A more restrictive formula used as a cost function tends to produce trajectories that closely resemble expert trajectories, often leading to lower overall costs. However, it's essential to consider that in such scenarios, there is a risk of overfitting the learned tree to the training data. This can result in a situation where the model does not allow for regions within the feature space which are not observed in the expert trajectories but not necessarily invalid. This could limit the agent's exploration and generalization capabilities and consequently may lead to

lower rewards. This reasoning is exemplified in Figure 4.4 where we depict the acquired reward and cost by agents trained on cost functions originating from trees with various depths.

4.4.2 Realistic Highway Environment

Traffic is a real-world environment where agent behavior is heavily governed by both explicit and implicit rules. We extract expert demonstrations from the highD dataset [5], a comprehensive repository of annotated vehicle trajectories recorded on German highways. To facilitate the training of Reinforcement Learning (RL) agents within this environment, we transform this dataset into multiple scenarios adhering to the CommonRoad framework [14]. Subsequently, we employ the CommonRoad-RL framework for agent training [15]. All the scenarios derived from the highD dataset encompass either two or three lanes in both directions. The feature space is characterized by the velocity along the x- and y-axes, the relative position of agents leading and following the ego-agent in the left, same and right lane, and the proximity to the left and right road edges $\{v_x, v_y, p_{\text{rel}_0}, p_{\text{rel}_1}, p_{\text{rel}_2}, p_{\text{rel}_3}, p_{\text{rel}_4}, p_{\text{rel}_5}, d_{\text{left}}, d_{\text{right}}\}$. The output of the distance sensor, which gauges the distance to surrounding vehicles, spans a range from 0 to 500. Our experiments span two distinct settings: one where an agent is trained on a restricted set of 52 scenarios from the highD dataset (i.e., highD simple) and another where an agent is trained on all 3000 scenarios (i.e., highD full). Due to the absence of ground truth constraints in this environment, it is not feasible to report the frequency of constraint violations. Instead, we present two key performance metrics: the off-road ratio (indicating the percentage of trajectories in which the agent deviates from the road) and the collision ratio (reflecting the percentage of trajectories in which the agent collides with another vehicle). For the purpose of comparison, our approach is evaluated against two other agent types: a standard RL agent without any provided constraints (i.e., RL) and an RL agent equipped with an engineered reward function, as proposed by Wang et al. [15], which issues negative rewards when the agent deviates off-road or collides with another vehicle (i.e., RL w. optimized reward). The results displayed in Figure 4.5 clearly demonstrate that our method excels in preventing agents from driving off-road and reducing the incidence of collisions with other vehicles. These outcomes strongly suggest that our rule-learned agents are not only safer but also achieve higher rewards compared to both the conventional RL agents and those equipped with the hand-engineered optimized reward function.

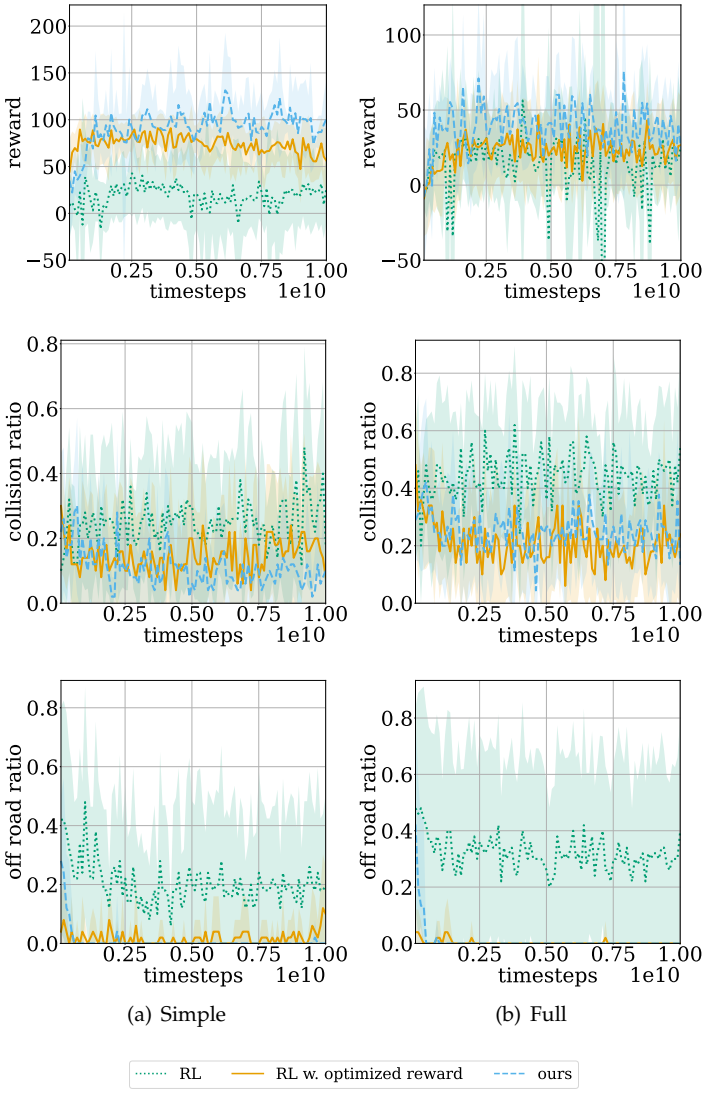


Figure 4.5: Reward (top), collision rate (middle) and off-road ratio (bottom) during training in a realistic highway environment.

4.5 Related Work

The constraint learning problem is by definition ill-posed as many set of constraints can describe the same set of expert trajectories. In order to prevent the acquisition of overly cautious constraints, many approaches rely on the principle of maximum entropy to derive the most concise set of constraints that align with expert demonstrations [16]. Scobee et al. [17] introduced an approach focused on learning a set of constraints that maximize the likelihood of expert demonstrations. This method, when combined with inductive logic programming, allows for the acquisition of a logical formula representing the learned constraints [18]. By approximating the maximum likelihood objective, this approach can be effectively applied to environments with continuous state-action spaces [19, 20, 21]. However, it is important to emphasize that these methods are primarily well-suited for environments with deterministic dynamics. For environments with stochastic dynamics, constraint learning is made possible through methods based on the principle of maximum causal entropy [22], as demonstrated by McPherson et al. [23] for discrete environments and by Baert et al. [13] for continuous environments. Unlike our methodology, the aforementioned methods all necessitate knowledge of the agent's goal and the availability of a simulator of the environment for learning the constraints. Of particular relevance to our research is the work of Lindner et al. [7], who also define constraints as a set in the environment's feature vector space. Nevertheless, their definition of the safe set involves the convex hull of the feature expectations derived from the demonstrations. In contrast, our approach models the safe set using a decision tree, offering the distinct advantage of interpretability for both the safe set and the constraints extracted. Additionally, we hypothesize that our method is less prone to overfitting, as it uses only the most informative feature dimensions for defining the safe set. Furthermore, it is important to note that they learn a linear cost function in feature space. To allow their method to work in environments with non-linear constraints, they adopt a one-hot encoding of the state-action space as features. The latter is only possible when the state-action space is relatively small and finite. Because of this, the method proposed by Lindner et al. [7] is unsuitable for learning constraints in the environments we evaluated. Another benefit of our method is that the OC-tree naturally provides robustness when dealing with a limited number of negative examples in the training data, a common scenario when learning from human demonstrations.

4.6 Conclusion

We have introduced a novel approach to acquire constraints modeled by a logic formula in disjunctive normal form from expert demonstrations. These acquired rules can be employed as an easily interpretable cost function in the context of constrained reinforcement learning. We evaluated our method on multiple synthetic environments and realistic autonomous driving scenarios. In both, our method improves safety and performance of the learning agents. In addition, we provided a post-training pruning mechanism to simplify to learned formula. Nevertheless, there exists substantial potential for enhancing the transferability of constraints across various domains. This improvement will alleviate the necessity for expert trajectories to be available in every target domain. In future work, we plan to investigate the relation between the tree depth and the transferability of the learned constraints. We would also like to investigate whether more effective constraints can be learned when a simulator of the environment is provided. At last, we want to extend the proposed method for learning constraints expressed in temporal logic. This extension brings the significant benefit of enhanced expressiveness, enabling us to effectively describe more intricate and complex constraints.

References

[1] S. Russell. *Human compatible: Artificial intelligence and the problem of control*. Penguin, 2019.

[2] V. Krakovna, J. Uesato, V. Mikulik, M. Rahtz, T. Everitt, R. Kumar, Z. Kenton, J. Leike, and S. Legg. *Specification gaming: the flip side of AI ingenuity*, 2020. Available from: https://deepmind.com/blog/article/Specification-gaming-the-flip-side-of-AI-ingenuity.

[3] S. Itani, F. Lecron, and P. Fortemps. *A one-class classification decision tree based on kernel density estimation*. Applied soft computing, 91:106250, 2020.

[4] A. Ray, J. Achiam, and D. Amodei. *Benchmarking safe exploration in deep reinforcement learning*. arXiv preprint arXiv:1910.01708, 7(1):2, 2019.

[5] R. Krajewski, J. Bock, L. Kloeker, and L. Eckstein. *The highD Dataset: A Drone Dataset of Naturalistic Vehicle Trajectories on German Highways for Validation of Highly Automated Driving Systems*. In 2018 21st International Conference on Intelligent Transportation Systems (ITSC), pages 2118–2125, 2018. doi:10.1109/ITSC.2018.8569552.

[6] E. Altman. *Constrained Markov decision processes: stochastic modeling*. Routledge, 1999.

[7] D. Lindner, X. Chen, S. Tschiatschek, K. Hofmann, and A. Krause. *Learning Safety Constraints from Demonstrations with Unknown Rewards*. arXiv preprint arXiv:2305.16147, 2023.

[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. *Proximal Policy Optimization Algorithms*. CoRR, abs/1707.06347, 2017. Available from: http://arxiv.org/abs/1707.06347, arXiv:1707.06347.

[9] J. Ji, J. Zhou, B. Zhang, J. Dai, X. Pan, R. Sun, W. Huang, Y. Geng, M. Liu, and Y. Yang. *OmniSafe: An Infrastructure for Accelerating Safe Reinforcement Learning Research*. arXiv preprint arXiv:2305.09304, 2023.

[10] J. Achiam, D. Held, A. Tamar, and P. Abbeel. *Constrained policy optimization*. In International conference on machine learning, pages 22–31. PMLR, 2017.

[11] T. Yang, J. Rosca, K. Narasimhan, and P. J. Ramadge. *Projection-Based Constrained Policy Optimization*. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net, 2020. Available from: https://openreview.net/forum?id=rke3TJrtPS.

[12] Y. Zhang, Q. Vuong, and K. Ross. *First order constrained optimization in policy space*. Advances in Neural Information Processing Systems, 33:15338–15349, 2020.

[13] M. Baert, P. Mazzaglia, S. Leroux, and P. Simoens. *Maximum Causal Entropy Inverse Constrained Reinforcement Learning*. arXiv preprint arXiv:2305.02857, 2023.

[14] M. Althoff, M. Koschi, and S. Manzinger. *CommonRoad: Composable benchmarks for motion planning on roads*. In 2017 IEEE Intelligent Vehicles Symposium (IV), pages 719–726. IEEE, 2017.

[15] X. Wang, H. Krasowski, and M. Althoff. *CommonRoad-RL: A Configurable Reinforcement Learning Environment for Motion Planning of Autonomous Vehicles*. In IEEE International Conference on Intelligent Transportation Systems (ITSC), 2021. doi:10.1109/ITSC48978.2021.9564898.

[16] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, et al. *Maximum entropy inverse reinforcement learning*. In Aaai, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[17] D. R. R. Scobee and S. S. Sastry. *Maximum Likelihood Constraint Inference for Inverse Reinforcement Learning*. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net, 2020. Available from: https://openreview.net/forum?id=BJliakStvH.

[18] M. Baert, S. Leroux, and P. Simoens. *Inverse reinforcement learning through logic constraint inference*. Machine Learning, pages 1–26, 2023.

[19] S. Malik, U. Anwar, A. Aghasi, and A. Ahmed. *Inverse constrained reinforcement learning*. In International conference on machine learning, pages 7390–7399. PMLR, 2021.

[20] A. Gaurav, K. Rezaee, G. Liu, and P. Poupart. *Learning Soft Constraints From Constrained Expert Demonstrations*. In The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net, 2023. Available from: https://openreview.net/pdf?id=8sSnD78NqTN.

[21] G. Liu, Y. Luo, A. Gaurav, K. Rezaee, and P. Poupart. *Benchmarking Constraint Inference in Inverse Reinforcement Learning*. In The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net, 2023. Available from: https://openreview.net/pdf?id=vINj_Hv9szL.

[22] B. D. Ziebart, J. A. Bagnell, and A. K. Dey. *Modeling Interaction via the Principle of Maximum Causal Entropy*. In J. Fürnkranz and T. Joachims, editors, Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel, pages 1255–1262. Omnipress, 2010. Available from: https://icml.cc/Conferences/2010/papers/28.pdf.

[23] D. L. McPherson, K. C. Stocking, and S. S. Sastry. *Maximum likelihood constraint inference from stochastic demonstrations*. In 2021 IEEE Conference on Control Technology and Applications (CCTA), pages 1208–1213. IEEE, 2021.

4.7 Appendix

4.7.1 Examples of Learned Rules

We present some learned formulas describing the constraints in an environment. The presented formulas are already pruned using a violation-evaluation threshold of 0.001.

PointGoal: The first rules define a maximum acceleration and velocity. The following rules define a safe boundary between the agent and hazards in different directions around the agent.

HighD: The first rules define a minimum and maximum speed limit along the x- and y-axis and a limit on the distance between the car in the same lane in front of the agent (p_{rel_4}) and behind the agent (p_{rel_1}). The next rules define a minimum distance to the road edges. The following rules distinguish the agent being on the left or right lane based on the distance to the right road edge. The last rule states the distance to the vehicle on the left lane in front of the agent should be smaller than 344.91 if the distance to the vehicle on the left lane behind the agent is bigger than 325.83. This will force the agent to move one lane to the left. It is debatable whether this last rule corresponds to desired behavior as the agent will have a preference to drive on the left or middle lane.

Rules extracted from the expert trajectories in the *PointGoal* environment after pruning.

$$
\begin{aligned}
\varphi = {} & a_x < -5.42 \\
& \vee v_y < -1.05 \\
& \vee v_x > 1.46 \\
& \vee d_{l0} > 0.88 \\
& \vee d_{l3} > 1.0 \\
& \vee d_{l5} > 0.99 \\
& \vee d_{l0} >= 0.0 \wedge d_{l0} <= 0.22 \wedge d_{l1} > 0.81 \\
& \vee d_{l0} >= 0.22 \wedge d_{l0} <= 0.88 \wedge d_{l7} > 0.87 \\
& \vee d_{l0} >= 0.0 \wedge d_{l0} <= 0.22 \wedge d_{l1} >= 0.0 \wedge d_{l1} <= 0.27 \wedge d_{l15} > 0.66 \\
& \vee d_{l0} >= 0.0 \wedge d_{l0} <= 0.22 \wedge d_{l1} >= 0.27 \wedge d_{l1} <= 0.81 \wedge d_{l15} > 0.81 \\
& \vee d_{l0} >= 0.22 \wedge d_{l0} <= 0.88 \wedge d_{l7} >= 0.0 \wedge d_{l7} <= 0.27 \wedge d_{l6} > 0.85 \\
& \vee d_{l0} >= 0.22 \wedge d_{l0} <= 0.88 \wedge d_{l7} >= 0.27 \wedge d_{l7} <= 0.87 \wedge d_{l14} > 0.94 \\
& \vee d_{l0} >= 0.0 \wedge d_{l0} <= 0.22 \wedge d_{l1} >= 0.0 \wedge d_{l1} <= 0.27 \wedge d_{l15} >= 0.0 \\
& \qquad \wedge d_{l15} <= 0.66 \wedge d_{l2} > 0.83 \\
& \vee d_{l0} >= 0.0 \wedge d_{l0} <= 0.22 \wedge d_{l1} >= 0.27 \wedge d_{l1} <= 0.81 \wedge d_{l15} >= 0.0 \\
& \qquad \wedge d_{l15} <= 0.81 \wedge d_{l14} > 0.94 \\
& \vee d_{l0} >= 0.22 \wedge d_{l0} <= 0.88 \wedge d_{l7} >= 0.0 \wedge d_{l7} <= 0.27 \wedge d_{l6} >= 0.0 \\
& \qquad \wedge d_{l6} <= 0.31 \wedge d_{l8} > 0.78 \\
& \vee d_{l0} >= 0.22 \wedge d_{l0} <= 0.88 \wedge d_{l7} >= 0.0 \wedge d_{l7} <= 0.27 \wedge d_{l6} >= 0.31 \\
& \qquad \wedge d_{l6} <= 0.85 \wedge d_{l8} > 0.89 \\
& \vee d_{l0} >= 0.22 \wedge d_{l0} <= 0.88 \wedge d_{l7} >= 0.27 \wedge d_{l7} <= 0.87 \wedge d_{l14} >= 0.0 \\
& \qquad \wedge d_{l14} <= 0.28 \wedge d_{l13} > 0.94 \\
& \vee d_{l0} >= 0.22 \wedge d_{l0} <= 0.88 \wedge d_{l7} >= 0.27 \wedge d_{l7} <= 0.87 \wedge d_{l14} >= 0.28 \\
& \qquad \wedge d_{l14} <= 0.94 \wedge d_{l4} > 0.94
\end{aligned}
$$

Rules extracted from the human trajectories in the highD dataset after pruning.

$$
\begin{aligned}
\varphi = & v_y < -0.95 \\
& \vee v_y > 0.96 \\
& \vee v_x < 20.67 \\
& \vee v_x > 47.47 \\
& \vee p_{\text{rel}_1} < 18.83 \\
& \vee p_{\text{rel}_4} < 18.83 \\
& \vee d_{\text{left edge}} < 4.42 \\
& \vee d_{\text{right edge}} < 2.02 \\
& \vee v_y >= -0.95 \wedge v_y <= 0.96 \wedge d_{\text{right edge}} >= 4.65 \wedge d_{\text{right edge}} <= 7.38 \\
& \qquad \wedge p_{\text{rel}_1} >= 18.83 \wedge p_{\text{rel}_1} <= 328.76 \wedge p_{\text{rel}_0} < 236.11 \\
& \vee v_y >= -0.95 \wedge v_y <= 0.96 \wedge d_{\text{right edge}} >= 4.65 \wedge d_{\text{right edge}} <= 7.38 \\
& \qquad \wedge p_{\text{rel}_1} >= 328.76 \wedge p_{\text{rel}_1} <= 500.0 \wedge p_{\text{rel}_0} < 155.9 \\
& \vee v_y >= -0.95 \wedge v_y <= 0.96 \wedge d_{\text{right edge}} >= 4.65 \wedge d_{\text{right edge}} <= 7.38 \\
& \qquad \wedge p_{\text{rel}_1} >= 18.83 \wedge p_{\text{rel}_1} <= 328.76 \wedge p_{\text{rel}_0} >= 236.11 \\
& \qquad \wedge p_{\text{rel}_0} <= 500.0 \wedge p_{\text{rel}_3} < 235.66 \\
& \vee v_y >= -0.95 \wedge v_y <= 0.96 \wedge d_{\text{right edge}} >= 4.65 \wedge d_{\text{right edge}} <= 7.38 \\
& \qquad \wedge p_{\text{rel}_1} >= 328.76 \wedge p_{\text{rel}_1} <= 500.0 \wedge p_{\text{rel}_0} >= 155.9 \\
& \qquad \wedge p_{\text{rel}_0} <= 500.0 \wedge p_{\text{rel}_3} < 136.43 \\
& \vee v_y >= -0.95 \wedge v_y <= 0.96 \wedge d_{\text{right edge}} >= 2.02 \wedge d_{\text{right edge}} <= 4.65 \\
& \qquad \wedge p_{\text{rel}_0} >= 0.04 \wedge p_{\text{rel}_0} <= 325.83 \\
& \qquad \wedge p_{\text{rel}_3} >= 0.03 \wedge p_{\text{rel}_3} <= 332.09 \wedge p_{\text{rel}_4} < 26.81 \\
& \vee v_y >= -0.95 \wedge v_y <= 0.96 \wedge d_{\text{right edge}} >= 2.02 \wedge d_{\text{right edge}} <= 4.65 \\
& \qquad \wedge p_{\text{rel}_0} >= 325.83 \wedge p_{\text{rel}_0} <= 500.0 \wedge p_{\text{rel}_1} >= 33.84 \\
& \qquad \wedge p_{\text{rel}_1} <= 219.6 \wedge p_{\text{rel}_4} < 29.48 \\
& \vee v_y >= -0.95 \wedge v_y <= 0.96 \wedge d_{\text{right edge}} >= 2.02 \wedge d_{\text{right edge}} <= 4.65 \\
& \qquad \wedge p_{\text{rel}_0} >= 325.83 \wedge p_{\text{rel}_0} <= 500.0 \wedge p_{\text{rel}_1} >= 219.6 \\
& \qquad \wedge p_{\text{rel}_1} <= 500.0 \wedge p_{\text{rel}_3} > 344.91
\end{aligned}
$$

Part II

Learning Plans from Demonstration

5

Learning Temporal Task Specifications From Demonstrations

"Beyond the walls of intelligence, life is defined."

Nas, "N.Y. State of Mind"

Learning Temporal Task Specifications From Demonstrations

M. Baert • S. Leroux • P. Simoens.
Presented at the International Workshop on Explainable, Transparent Autonomous Agents and Multi-Agent Systems (EXTRAAMAS), 2024.

This chapter tackles the challenge of acquiring task specifications in linear temporal logic through expert demonstrations, aiming to alleviate the burdensome task of specification engineering. The rich semantics of temporal logics serve as an interpretable framework for delineating intricate, multi-stage tasks. We propose a method which iteratively learns a task specification and a nominal policy solving this task. In each iteration, the task specification is refined to better distinguish expert trajectories from trajectories sampled from the nominal policy. This process bridges the gap between intuitive task demonstrations and formal task representations, making the design of task specifications both more accessible and accurate. Unlike previous work, our method is capable of learning directly from trajectories in the original state space and does not require predefined atomic propositions. We showcase the effectiveness of our method on multiple tasks in both an office and a Minecraft-inspired environment.

5.1 Introduction

Consider an autonomous agent tasked with solving a multi-stage objective, wherein a sequence of subtasks must be executed in a particular order. Besides achieving the final goal, the agent should also respect various constraints. An example of such a task is illustrated in Figure 5.1, where the environment challenges the agent to craft a stone tool. To accomplish this, the agent must initially acquire wood and stone, before he can proceed to the task of building the tool at the crafting bench. Furthermore, the agent must avoid encounters with hostile monsters. In the realm of Reinforcement Learning (RL), conventional reward functions are not suitable for solving tasks with temporal dependencies because the reward signal is assumed to have the Markov property (i.e., the reward signal only depends on the current state and action). In contrast, it has been shown that temporal logics, such as Linear Temporal Logic (LTL) [1], offer an effective language for encoding non-Markovian reward functions [2, 3, 4, 5]. LTL is an expressive formal language that integrates temporal modalities like *until* and *globally* with Boolean propositions determining the truth or falsity of an event or property. These are interconnected through logical connectives, facilitating the specification of goal sequences, partial-order tasks, safety constraints, and various other conditions in an interpretable manner. For the crafting

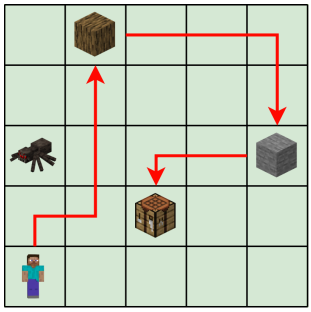


Figure 5.1: Minecraft inspired environment, adapted from Andreas et al. [7]

example depicted in Figure 5.1, the task can be specified by the following LTL formula,

$$\varphi = G(!\,\text{🕷}) \wedge (\text{🪵}\,\mathcal{T}\,\text{🪵}) \wedge (\text{🪨}\,\mathcal{T}\,\text{🪵}). \tag{5.1}$$

This formula states the agent should always avoid the spider (🕷) and should first acquire wood (🪵) and stone (🪨) before going to the work bench (🪵). Usually, a domain expert is responsible for specifying the LTL formula, but this is a complex and error-prone process for real-world applications. This is problematic as imperfect rewards can be exploited by a learning agent (i.e., reward hacking), leading to algorithmically correct but unintended high-gain solutions [6].

Since it is often much easier to demonstrate a task than to accurately define the objective, we propose a methodology for learning a task specification in LTL from observations. We start with a randomly initialized nominal policy and a collection of expert trajectories. Subsequently, we learn an LTL formula to distinguish between expert trajectories and those generated by the random policy. We then update the nominal policy by training it on the inferred formula. This alternation between updating the objective and learning the optimal policy continues until the nominal policy aligns with the unknown expert policy. Returning to the crafting example, in the initial iteration, we might infer the objective as $\varphi = F(\text{🪵})$, indicating the agent should eventually fetch some wood. This formula effectively distinguishes the nominal trajectories (which at this point are random walks) from the expert trajectories because it holds true for all expert trajectories while most nominal trajectories will not satisfy this formula. Subsequently, the nominal policy is optimized based on this objective and will produce in the next iteration trajectories that correspond with a path from the initial position to the wood location. In the next iteration, our objective could potentially extend to $\varphi = F(\text{🪵}) \wedge F(\text{🪵})$. Each iteration involves expanding

our objective with elements that continue to differentiate nominal from expert trajectories. Training the nominal policy on this evolving objective refines the nominal trajectories in the subsequent iteration, making them more closely resemble expert trajectories.

In contrast to prior approaches relying on tightness [8] or conciseness [9, 10] measures, our method considers the goal-directed nature of the expert, ensuring only essential elements are included in the objective. This results in learned formulas that are more concise and interpretable compared to existing methods. These traditional approaches also necessitate that observations are represented as a set of Boolean propositions. However, trajectories are commonly defined in a high-dimensional state space, rendering these methods impractical. Our approach addresses this limitation by also learning a mapping from states to Boolean propositions. In summary, our method improves on the state-of-the-art by providing more interpretable formulas and enabling learning from trajectories defined in the environment's state space without relying on a predefined set of atomic propositions. The remainder of this chapter is organized as follows. Section 5.2 provides preliminaries on LTL, RL and reward machines. In Section 5.3 we discuss related work on RL with temporal logic rewards and learning temporal logics from data. Next, we present our method in Section 5.4. We provide a qualitative and quantitative evaluation of our method in Section 5.5. We conclude in Section 5.6.

5.2 Background

5.2.1 Linear Temporal Logic

Linear temporal logic (LTL) extends propositional logic with two temporal operators: the *Next* operator X and the *Until* operator U. An LTL specification φ is constructed from a finite set of atomic propositions AP, temporal and logical operators. The minimal syntax of LTL is defined as

$$\varphi := p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid X\varphi \mid \varphi_1 U \varphi_2, \tag{5.2}$$

where $p \in \text{AP}$. Unlike propositional logic, LTL formulas are evaluated over sequences of observations $[\mathbf{p}]$ where each observation corresponds with a truth valuation of variables $\mathbf{p} \in \text{AP}^n$ with n the cardinality of $\mathbf{p}$. The notation $[\mathbf{p}] \models \varphi$ indicates that $[\mathbf{p}]$ satisfies φ. The next operator $X\varphi$ is satisfied if φ holds the next timestep and the until operator $\varphi_1 U \varphi_2$ holds if φ_1 holds at least until φ_2 becomes true, which must hold at the current or a

future time step. From the base operators, we can derive

$$\begin{aligned}
\top &= p \vee \neg p \\
\bot &= \neg \top \\
\varphi_1 \wedge \varphi_2 &= \neg((\neg\varphi_1) \vee (\neg\varphi_2)) \\
F\varphi &= \top U \varphi \\
G\varphi &= \neg(F(\neg\varphi)) \\
\varphi_1 \mathcal{T} \varphi_2 &= F(\varphi_1 \wedge XF(\varphi_2)).
\end{aligned}$$

The temporal operator *Finally* $F\varphi$ holds if φ eventually becomes true and the *Globally* operator $G\varphi$ evaluates true if φ evaluates true for the complete sequence. The *Then* operator $\varphi_1 \mathcal{T} \varphi_2$ states that φ_1 should become true before φ_2 becomes true.

5.2.2 Reinforcement Learning

A Markov decision process (MDP) [11] is a mathematical framework for modeling sequential decision-making. An MDP is characterized by a state space $\mathcal{S}$, an action space $\mathcal{A}$, a discount factor $\gamma \in [0, 1]$, a transition distribution $p(s' \mid s, a)$ defining the probability of transitioning to state s' when taking action a from state s, an initial state distribution $\mathcal{I}(s)$, and a reward function $R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, specifying the scalar reward the agent receives for applying action a in state s. The agent interacts with the environment at discrete timesteps t, generating a path through the state space, called a trajectory $\tau = (s_0, ..., s_{T-1})$ of length T. The reward of a trajectory τ of length T is defined as the sum of discounted rewards: $R(\tau) = \sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)$. At each timestep, the agent's action is chosen based on a policy $\pi : \mathcal{S} \mapsto \mathcal{P}(\mathcal{A})$, mapping states to probability distributions over actions. The objective of RL is to determine the policy that maximizes the expected sum of discounted rewards: $\max_\pi \mathbb{E}_{\tau\sim\pi} R(\tau)$.

5.2.3 Reward Machines

In an MDP, the reward at a given time step is solely dependent on the current state and action. Consequently, the formulation of objectives with temporal dependencies becomes impossible within this framework. To broaden the scope of MDPs, Reward Machines (RMs) offer a powerful framework for representing non-Markovian objectives. Given a set of atomic propositions AP, a set of environment states $\mathcal{S}$, and a set of actions $\mathcal{A}$, a Reward Machine (RM) is defined as a tuple $\mathcal{R}_{\text{AP},\mathcal{S},\mathcal{A}} = \langle U, u_0, \delta_u, \delta_r \rangle$. Here, U is a finite set of states, $u_0 \in U$, represents the initial state, δ_u is the state-transition function given by $\delta_u : U \times \text{AP} \to U$, and δ_r is the reward-transition function defined as $\delta_r : U \times U \to \mathbb{R}$. At each time step t, the RM takes a truth assignment $\mathbf{p}_t$

as input. This assignment is a set that precisely includes propositions in AP that are true in the environment state s_t. In this work we will only consider simple RM's meaning that $\delta_r(u, u')$ is constant for all states $u \in U$. We can replace the standard reward in a Markov Decision Process (MDP) by an RM (i.e., MDPRM). The only requirement is the availability of a labeling function $L : \mathcal{S} \to 2^{|\text{AP}|}$ which assigns truth values to the symbols in AP given an environment state. The state of the RM is updated every time step of the environment. If the RM is in state u and the agent performs action a to move from environment state s to s', the RM moves to state $u' = \delta_u(u, L(s'))$ and the agent receives reward $r = \delta_r(u, u')$. A policy $\pi(a \mid s, u)$ for an MDPRM defines a probability distribution over actions conditioned on the environment state s and the RM state u. Because of this, it is possible to model some non-Markovian reward functions in an MDPRM as long as different environment state histories can be distinguished by different elements of a finite set of regular expressions over AP. RM's can thus return different rewards for the same environment transitions (s, a, s'), for different histories of states seen by the agent.

5.3 Related Work

5.3.1 Reinforcement Learning with Temporal Logic Rewards

Many approaches to learning from demonstration characterize the problem as one of acquiring a standard Markovian reward function [12, 13, 14]. Nevertheless, the utility of these reward functions is constrained to short-horizon tasks. To broaden the scope of learning from demonstration and tackle more intricate tasks, significant effort has been invested in exploring more expressive models, such as temporal logics, to define the reward signal. Early work uses the robustness degree as a quantitative semantics of temporal logics [15, 2]. Xiong et al. [16] propose a differentiable quantitative Signal Temporal Logic (STL) semantics enabling the use of gradient based methods for optimizing the objective. Recurrent neural networks provide a natural way of encoding temporal dependencies as shown by Kuo et al. [17]. However, this involves solving a partially observable MDP which has no convergence guarantees. Another line of work, converts the non-Markovian reward which is typical for temporal tasks to a Markovian reward by augmenting the state space with information about the past [18, 19]. This forms the basic principle of reward machines [19] which are used in this work.

5.3.2 Inferring Temporal Logic from Data

Because manually crafting LTL formulas requires expert knowledge and is error-prone, many works try to infer LTL formulas from data. Most methods

rely on a labeled dataset consisting of positive and negative traces [20, 21]. Approaches based on decision trees, as proposed by Kong et al. [21] and Bombara et al. [20], aim to learn both operators and parameters by expanding logic branches. These methods use the robustness metric for assessing how much a trajectory satisfies a formula. Another line of work uses neural networks to represent STL formulas [22, 23, 24, 25]. Because the original LTL robustness metric is not differentiable, these methods propose different differentiable approximations of this metric. Although, the parameters of the STL formulas can be efficiently learned by back-propagation, the formula structure is fixed by the network architecture. Yan et al. [26] propose a neuro-symbolic approach combining the tree-based and neural approach such that both structure and parameters can be learned efficiently. However, all these methods assume the availability of both positive and negative examples while in many cases collecting negative examples is not straightforward (e.g., in a self-driving scenario it would be dangerous to collect examples of cars ignoring traffic lights). Some recent works propose methods for learning LTL from solely positive examples [9, 10]. Shah et al. [9] take a Bayesian approach and learn a distribution over LTL formulas. Roy et al. [10] convert the task of learning an LTL formula into a series of Boolean satisfiability problems. Vazquez-Chanlatte et al. [27] find inspiration in the MaxEnt framework as they formulate their method, which involves leveraging demonstrations to compute the posterior probability of an agent endeavoring to fulfill a Boolean specification. In contrast to our approach, these methods assume that for each time step in the provided trajectories, the truth valuation of a set of Boolean propositions is given. On the other hand, Jha et al. [8] and Chiu et al. [28] allow learning STL formulas from continuous observations but are limited to very simple formulas. Closest to our work is the work of Chou et al. [29], as their method is also capable of learning both the formula structure and the mapping from states to atomic propositions. The main difference, however, is that their solution builds on integer programming and requires complete knowledge of the environment's dynamics. In contrast, our method is based on interaction with the environment and considers its dynamics as a black box. A similar line of work, focuses on learning reward machines from demonstrations [30, 31, 32]. However, reward machines are inherently less interpretable compared to LTL formulas. Moreover, these methods also require the set of atomic propositions to be predefined.

5.4 Method

We propose an iterative procedure to learn a task description from a set of expert trajectories $\mathcal{D}_E$. Our method alternately refines this task specification

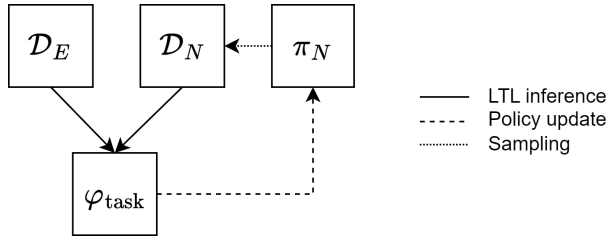


Figure 5.2: LTL inference - policy update loop

represented by an LTL formula φ^i_{task} and learns a policy π^i_N optimizing this specification. We refer to this policy as the nominal policy. The superscript i denotes the training iteration. Figure 5.2 visualizes the iterative training process. The initial task formula is defined as $\varphi^0_{\text{task}} = \top$ and the nominal policy is initialized randomly. In each iteration an LTL formula φ^* is inferred that distinguishes the expert demonstrations $\mathcal{D}_E$ from trajectories sampled from the nominal policy $\mathcal{D}_N$ (i.e., nominal trajectories). Next, we extend the task specification φ^i_{task} with the inferred formula φ^* such that $\varphi^i_{\text{task}} = \varphi^{i-1}_{\text{task}} \cap \varphi^*$. Following, the nominal policy π^i_N is trained on the updated task description φ^i_{task}. Until a stopping criterion is met, we alternate between LTL inference and policy update. First we describe the formula hypothesis space. Next, both the LTL inference and policy update steps are described. At last, we specify the stopping criteria and a pruning mechanism for simplifying learned formulas.

5.4.1 Formula Hypothesis Space

Numerous tasks, encompassing various subtasks, can be effectively characterized through the integration of three temporal behaviors identified by Dwyer et al. [33]: global satisfaction of a proposition, eventual completion of a subtask, and temporal ordering between subtasks. These behaviors are captured by the *Globally*, *Finally* and *Then* operators, respectively. Our hypothesis space for selecting a single temporal operator is restricted to this finite set of LTL templates and their negation. After selecting a template, it is instantiated with a selection of n_T propositions represented by $\mathbf{p} \in \text{AP}^{n_T}$. Here, n_T represents the arity of the template, the *Globally* and *Finally* operators have arity 1 and the *Then* operator has arity 2. For examples, an instantiation of $T = \{G\}$ with $\mathbf{p} = [a]$ is written as $G(a)$. The expert demonstrations are represented as observed sequences of state variables, $s \in \mathcal{S}$. Since LTL formulas are defined over Boolean propositions, we require a mapping which assigns truth values to the symbols in AP given an environment state. In this work, we consider both the case where AP and

the corresponding mapping are given and the case they should be learned.

5.4.2 LTL Inference

Given a set of expert demonstrations $\mathcal{D}_E$ and a nominal policy π_N, we want to learn an LTL formula φ^* which distinguishes behavior belonging to the expert from behavior belonging to the nominal agent. First, we build a dataset of nominal trajectories $\mathcal{D}_N$ through sampling the nominal policy. A labeled dataset is constructed from the expert and nominal trajectories: $\mathcal{D} = \mathcal{D}_E \cup \mathcal{D}_N$. Next, we want to learn an LTL formula φ^* which discriminates between the nominal and the expert trajectories, this is the instantiated template which maximizes the information gain (IG):

$$\varphi^* = \underset{T \in \mathcal{T},\, \mathbf{p} \in \mathcal{H}_{\text{AP}}^{n_T}}{\text{argmax}} \ \text{IG}(T(\mathbf{p}), \mathcal{D}), \tag{5.3}$$

with $\mathcal{T}$ denoting the set of formula templates and $\mathcal{H}_{\text{AP}}$ representing the proposition hypothesis space. We define the information gain given an LTL formula φ and a labeled dataset $\mathcal{D}$ as

$$IG(\varphi, \mathcal{D}) = H(\mathcal{D}) - p_\top \cdot H(\mathcal{D}_\top) - p_\bot \cdot H(\mathcal{D}_\bot), \tag{5.4}$$

with

$$p_\top = \frac{\mid \mathcal{D}_\top \mid}{\mid \mathcal{D} \mid}, p_\bot = \frac{\mid \mathcal{D}_\bot \mid}{\mid \mathcal{D} \mid}, p_E = \frac{\mid \mathcal{D}_E \cap \mathcal{D} \mid}{\mid \mathcal{D} \mid}, p_N = \frac{\mid \mathcal{D}_N \cap \mathcal{D} \mid}{\mid \mathcal{D} \mid} \tag{5.5}$$

and $\mathcal{D}_\top = \{(\tau, l) \in \mathcal{D} \mid L(\tau) \models \varphi\}$, $\mathcal{D}_\bot = \{(\tau, l) \in \mathcal{D} \mid L(\tau) \not\models \varphi\}$. In other words, $\mathcal{D}_\top$ and $\mathcal{D}_\bot$ are subsets of the original dataset $\mathcal{D}$ for which φ is satisfied and not satisfied respectively. $p_\top$ and $p_\bot$ denote the fraction of $\mathcal{D}$ belonging to $\mathcal{D}_\top$ and $\mathcal{D}_\bot$ respectively. p_E and p_N denote the fraction of $\mathcal{D}$ corresponding to expert trajectories and nominal trajectories respectively. The entropy H of a dataset $\mathcal{D}$ is defined as

$$H(\mathcal{D}) = -p_E \log(p_E) - p_N \log(p_N). \tag{5.6}$$

If the set of atomic propositions AP is given, for some template T, the hypothesis space is defined as $\mathcal{H}_{\text{AP}} = \text{AP}$.

When the set of atomic propositions AP is unknown, we consider an initial empty set of atomic propositions $\text{AP} = \emptyset$ and the hypothesis space becomes $\mathcal{H}_{\text{AP}} = \mathcal{S}$. However, this implies each state is a potential proposition, and we should iterate n_T times over the entire state space $\mathcal{S}$ for each template T. To increase efficiency, we can significantly prune the hypothesis space by considering only states visited by the expert and nominal agents. This approach aligns with the logic that a task specification comprises states to

Algorithm 5.1: LTL inference

Input: $\mathcal{D}, \mathcal{T}$

$c^* = \infty$
$T^* = \text{null}$
$\mathbf{p}^* = \text{null}$
for $T \in \mathcal{T}$ **do**
  $\mathbf{p} = \arg\max_{\mathbf{p} \in \mathcal{H}_{AP}^{n_T}} \text{IG}(T(\mathbf{p}), \mathcal{D})$
  $c = \text{IG}(T(\mathbf{p}), \mathcal{D})^{-1} - 1$
  if $c < c^*$ **then**
    $c^* = c$
    $T^* = T$
    $\mathbf{p}^* = \mathbf{p}$
  end
end
return $T^*(\mathbf{p}^*)$

be visited and states to be avoided. Expert trajectories naturally encompass states that should be visited, while nominal trajectories include states that should be avoided. If a state is frequently visited by the nominal agent but never by the expert, it indicates the presence of a constraint that the expert considers but is absent from the current LTL task description. The LTL formula encoding this constraint will yield high information gain. Once a new formula $\varphi^* = T^*(\mathbf{p}^*)$ is learned, $\mathbf{p}^*$ is added to AP and maintain a map between the atomic propositions AP and the MDP's state space $\mathcal{S}$.

Algorithm 5.1 outlines the procedure for inferring the instantiated template $\varphi^* = T^*(\mathbf{p}^*)$ which maximized the information gain based on a dataset of expert and nominal trajectories $\mathcal{D}$ and a set of formula templates $\mathcal{T}$. Subsequently, we expand our task description by updating it to $\varphi^i_{\text{task}} = \varphi^{i-1}_{\text{task}} \cap \varphi^*$.

5.4.3 Policy update

During the policy update step, we learn a policy which maximizes the up till now inferred task specification φ^i_{task}. As LTL formulas are by definition non-Markovian, they cannot be used as a reward function in standard RL approaches. However, it has been shown that LTL formulas can be automatically translated into RMs [34]. We use Spot [35] to first translate the inferred LTL formula φ^i_{task} into a complete state-based Büchi automaton [36]. To translate the Büchi automaton to an RM, a constant reward of

1 is assigned to transitions to one of the automaton's accepting states. In other words, $\delta_r(u, u')$ is 1 if u' is an accepting state in the Büchi automaton and zero otherwise. Icarte et al. [19] propose multiple practical implementations for learning policies optimizing an RM. We adopt the Counterfactual Experiences for Reward Machines (CRM) method in combination with Q-learning. In this method a policy is learned over the cross-product of the environment's state space $\mathcal{S}$ and the RM's state space U: $\pi(a \mid s, u)$ and uses counterfactual reasoning to generate synthetic experiences.

5.4.4 Stopping conditions

When learning only from positive examples, two challenges arise. The first one is overgeneralization, where the learned formula is excessively broad. For instance, the formula $\varphi = \top$ encompasses all trajectories, but it is overly general. While being the most concise model that accepts all expert trajectories, it lacks specificity. The second problem is overfitting, where the inferred formula is overly stringent. Stopping our method prematurely may produce a formula that is too general. Conversely, running the method for an extended period could lead to a formula that is overly specific. To address this, we require a metric to determine when to terminate the learning process. Here, we use again the information gain. The information gain is a measure of the reduction in entropy achieved by splitting a dataset. The information gain is equal to one for formulas that satisfy all expert trajectories while rejecting all nominal trajectories. If a formula fails to satisfy all expert trajectories, it is likely not a crucial aspect of the task. Additionally, if a formula satisfies some nominal trajectories and some expert trajectories, it is improbable that this formula accurately describes a constraint. If the information gain of the inferred formula φ^* falls below a predefined threshold, we terminate the learning process, and the last inferred formula φ^* is not added to φ^i_{task}. This serves as a mechanism to avoid incorporating overly general or overly specific formulas into the learned task description.

5.4.5 Formula Simplification

Because we adopt a greedy approach for assembling the task specification it is probable that some parts of the resulting task specification are redundant. Some subformulas learned during training serve as a guidance towards the real goal. Once the real goal or constraint is learned we can discard sub-goals which are inevitably achieved while pursuing the real goal. To this end, we propose a method for pruning LTL formula. We iterate over all subformulas and create RMs where for each RM one of the subformulas is held-out. For each RM we learn the corresponding optimal policy and sample trajectories (as described above). Then we measure the empirical

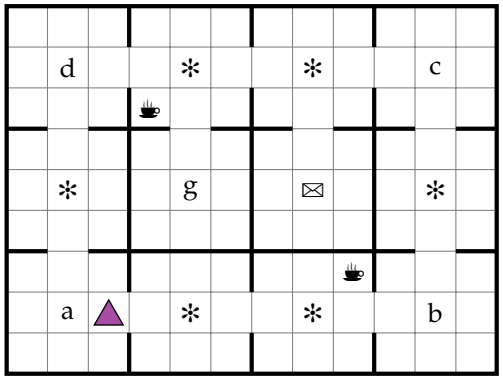


Figure 5.3: Office environment, originally proposed by [19]. The triangle represents the initial state of the agent.

probability the held out subformula is satisfied by trajectories sampled from the policy learned from the RM where the subformula is held out. If this probability is above some threshold, we can assume the learned subformula is redundant as some other formula causes the satisfaction of the held out subformula.

5.5 Results

We evaluate our method on multiple tasks in two gridworld environments with non-Markovian reward functions: the *office* environment [19] and the *craft* environment [7, 19]. The office environment is defined by a 12 by 9 grid containing multiple rooms and specific locations of interest which correspond with the ground truth set of atomic propositions (see figure 5.3). These locations include mail (✉), coffee (☕), decorations (✻) and offices (a, b, c, d, g). The craft environment is defined by a 40 by 40 grid where resources (a, b, c, d, e, f, g) are scattered around the map (see Figure 5.1 for a simplified version of this environment). Each environment comes with 4 tasks consisting of multiple subtasks and constraints. In both environments the agent can move in the four cardinal directions but not diagonally. Table 5.1 reports the ground truth task specifications in LTL for the four tasks for each environment, here *Office-N* refers to the N^{th} task in the office environment. We report results for our method with and without the formula simplification procedure presented in Section 5.4.5. We compare against two baseline methods for learning LTL formulas given only positive examples. The first one is based on the method of Kasenberg et al. [37] which infers an LTL formula in order to distinguish random behavior from the expert trajectories. To do this, we simply skip the policy update step in our method

Table 5.1: Ground truth task specifications in LTL

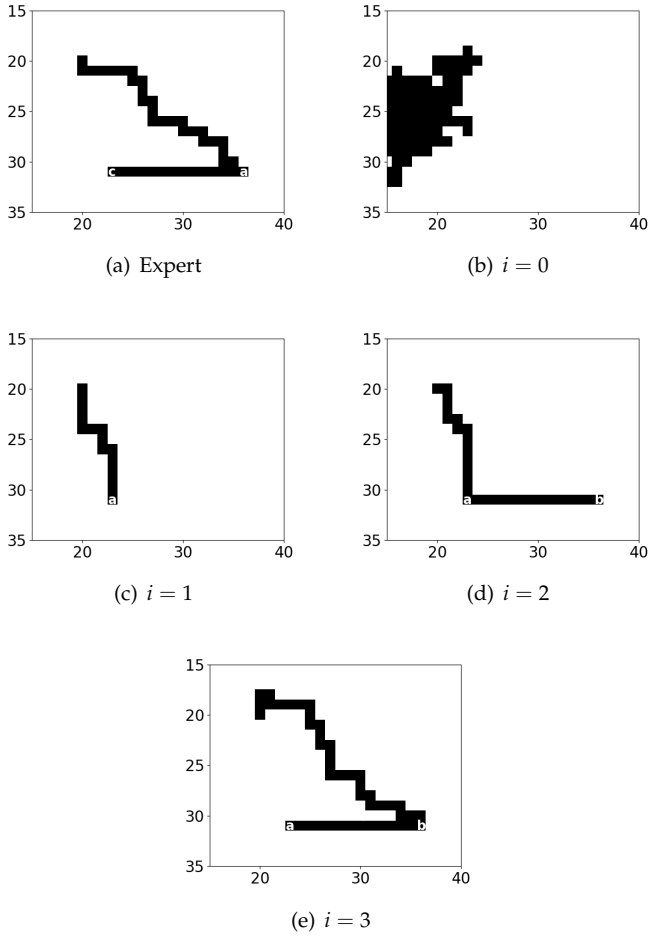
Environment-Task	φ_{task}
Office-1	$(☕\,\mathcal{T}\,g) \wedge G(!*)$
Office-2	$(✉\,\mathcal{T}\,g) \wedge G(!*)$
Office-3	$(☕\,\mathcal{T}\,g) \wedge (✉\,\mathcal{T}\,g) \wedge G(!*)$
Office-4	$(a\,\mathcal{T}\,b) \wedge (b\,\mathcal{T}\,c) \wedge (c\,\mathcal{T}\,d) \wedge G(!*)$
Craft-1	$(a\,\mathcal{T}\,c)$
Craft-2	$(a\,\mathcal{T}\,e) \wedge (f\,\mathcal{T}\,e)$
Craft-3	$(a\,\mathcal{T}\,b) \wedge (f\,\mathcal{T}\,b) \wedge (c\,\mathcal{T}\,b)$
Craft-4	$(a\,\mathcal{T}\,e) \wedge (f\,\mathcal{T}\,e) \wedge (e\,\mathcal{T}\,g)$

such that the nominal policy stays random. The second baseline is the method of Roy et al. [10], whose objective is based on formula conciseness. They frame their objective as multiple Boolean satisfiability problems and adopt a standard SAT-solver for obtaining the solution. For both baseline methods, we also convert the learned LTL formula to an RM and also use CRM Q-learning for acquiring the corresponding optimal policy. We ran experiments with the method of Jha et al. [8] but did not obtain good results, which suggests that their method is limited to simple STL formulas. We consider the case where AP is given and AP should be learned. In Subsection 5.5.1 we report the evolution of the nominal policy and the task specification during training as a qualitative example. In Subsection 5.5.2 we report quantitative measures which assess the nominal policy in solving the ground truth task, the bias in the nominal policy and the length of the learned formulas. At last, we assess the influence of the number of expert demonstrations on these quantitative measures in Subsection 5.5.3. All expert demonstrations and an implementation of our method and of the benchmark methods are available online.[1]

5.5.1 Qualitative Example

Table 5.2 reports the ground truth task specification and the learned specification after i iterations for the *Craft-1* task. Figure 5.4 shows one expert trajectory and the evolution of the nominal policy. For $i = 0$, the task specification and the set of atomic propositions are empty, and the nominal policy generates random behavior. During the first iteration, our algorithm learns

[1] https://gitlab.ilabt.imec.be/mwbaert/l-ltl-fd



(a) Expert (b) $i = 0$

(c) $i = 1$ (d) $i = 2$

(e) $i = 3$

Figure 5.4: Evolution of the nominal trajectories for the Craft-1 example with AP unknown. i represents the number of performed iterations. Table 5.2 reports the corresponding φ^i_{task} and AP.

Table 5.2: Evolution of the task specification φ^i_{task} and AP during training for the Craft-1 environment with AP unknown. The first row reports the ground truth task specification, the next lines correspond with the result after i iterations. The last row presents the task specification after pruning. Figure 5.4 depicts a trajectory sampled from the corresponding nominal policies.

	φ_{task}	AP
ground truth (Fig. 5.4(a))	$a \mathcal{T} c$	$\{a = (31, 36), c = (31, 23), ...\}$
$i = 0$ (Fig. 5.4(b))	$\top$	$\emptyset$
$i = 1$ (Fig. 5.4(c))	$F(a)$	$\{a = (31, 23)\}$
$i = 2$ (Fig. 5.4(d))	$F(a) \wedge F(b)$	$\{a = (31, 23), b = (31, 36)\}$
$i = 3$ (Fig. 5.4(e))	$F(a) \wedge F(b) \wedge (b \mathcal{T} a)$	$\{a = (31, 23), b = (31, 36)\}$
$i = 3$ (pruned)	$b \mathcal{T} a$	$\{a = (31, 23), b = (31, 36)\}$

one proposition $a = (31, 23)$ which should eventually become true. Note that we report both the proposition and the corresponding state. During the second iteration, our method discovers that $b = (31, 36)$ should eventually be satisfied. During the third iteration our method discovers the temporal ordering in satisfying these propositions. Note that the learned propositions' names are determined by when they are discovered. Because of this, the learned propositions will likely have different names then the ground truth propositions, this is also the case for the example in table 5.2 and figure 5.4.

5.5.2 Quantitative Evaluation

To evaluate the effectiveness of the recovered nominal policy, we quantify the success rate of each method on the ground truth task. To accomplish this, we deterministically select the optimal action in each state according to the nominal policy. To assess overfitting to the demonstrations, we measure the expected entropy of the learned policies, denoted as

$$H(\pi) = \mathbb{E}_{\tau \sim \pi,\, s_t, a_t \sim \tau} \left[H(\pi(a_t \mid s_t)) \right]. \tag{5.7}$$

The entropy of a policy measures the randomness of this policy, higher values indicate lower bias, which is preferable. Throughout all experiments, we use an information gain threshold of 0.2 as the stopping criteria and set the maximum number of iterations to 10. Results are averaged over 5 runs. For each environment we provide 50 expert trajectories. To obtain these trajectories we sample an expert policy trained on the ground truth task specification.

Figure 5.5 presents the results when we consider the definition of the atomic

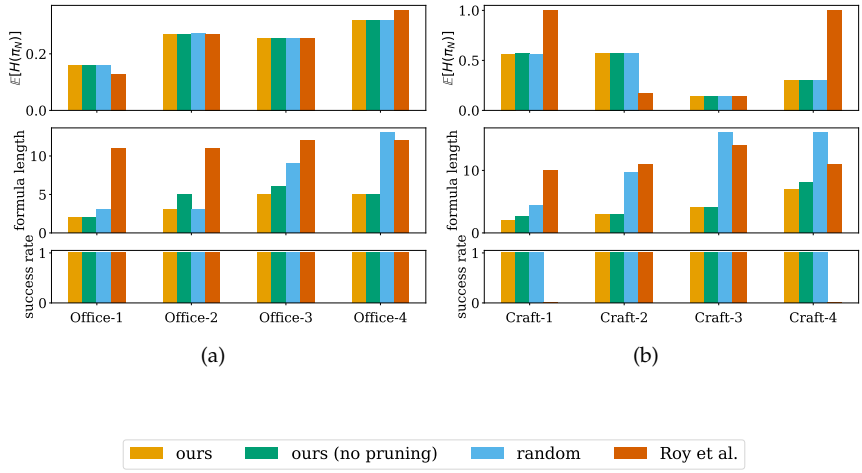


Figure 5.5: Quantitative evaluation of the learned task specification and the nominal policy with AP given.

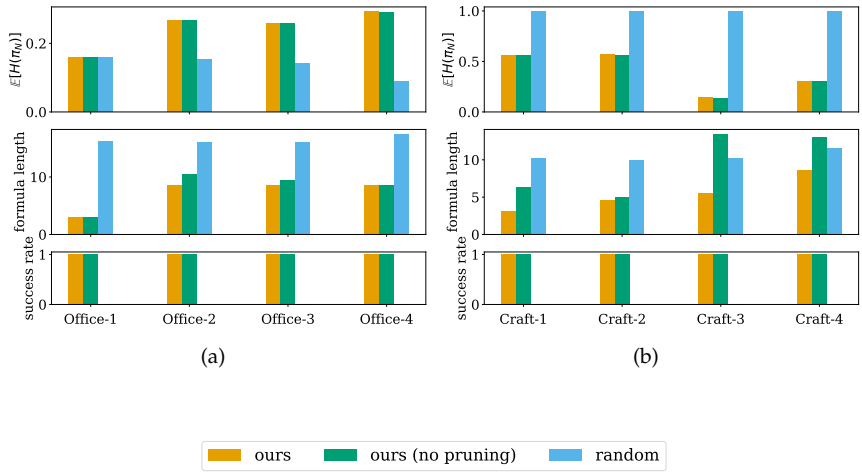


Figure 5.6: Quantitative evaluation of the learned task specification and the nominal policy with AP unknown.

propositions, AP, is given. Regarding the entropy of the nominal policy, longer formulas occasionally yield more biased policies, but this is not universally true. The value of the expected entropy also heavily depends on the task, as executions of some tasks which require more time steps (e.g., Office-4) are likely more diverse than executions of simple tasks (e.g., Office-1). Our method consistently recovers the shortest task specification. The method without policy updates (*random*) results in formulas with similar lengths as our method for simple tasks such as Office-1, Office-2 and Craft-1. However, for more complex tasks, the formula lengths are significantly higher for the method without policy update compared to ours. The method of Roy et al. [10] consistently results in large formulas which are harder to interpret. For the Craft-1 and Craft-4 environments, policies trained on task specifications from the method of Roy et al. [10] fail to solve these tasks. We attribute this to the task specification invalidating all propositions not observed by the expert, hindering exploration and preventing the discovery of the goal. In such cases, the policy does not converge, remaining mostly random and causing high entropy. Our method and the method without policy updates achieve the maximum success rate in all environments. It is noteworthy that formula lengths can be shorter than the ground truth while still achieving perfect success rates. This phenomenon can be attributed to the construction of the environment. For instance, in a 1-dimensional environment with propositions $a = 2, b = 4, c = 5$, the formula $\varphi_{\text{task}} = F(a) \wedge F(c)$ yields the same behavior as $\varphi_{\text{task}} = F(a) \wedge F(b) \wedge F(c)$.

Figure 5.6 illustrates results for unknown atomic propositions. When atomic propositions are unknown, the expanded hypothesis space for searching a task specification makes the task more challenging. In this scenario, only our method succeeds in solving all ground truth tasks. Other methods, not designed for such a large hypothesis space, fall short. We do not report results for the method of Roy et al. [10] as the method becomes intractable for the large hypothesis space. For the method without policy updates, long formulas are inferred as the model struggles to discern crucial parts of the task specification causing low success rates. Long formulas result in large RMs, which may prevent convergence to a successful policy because of the complexity of the task and the sparse reward. This is the case for the Craft environments for the method without policy updates as the entropy of the corresponding policies is ± 1, meaning the policy mostly random. When AP is not given, there is often a larger gap between our method with and without formula pruning. We attribute this to larger formula hypothesis space. Because of this, it is more likely to infer redundant subformulas which can be removed from the final task specification.

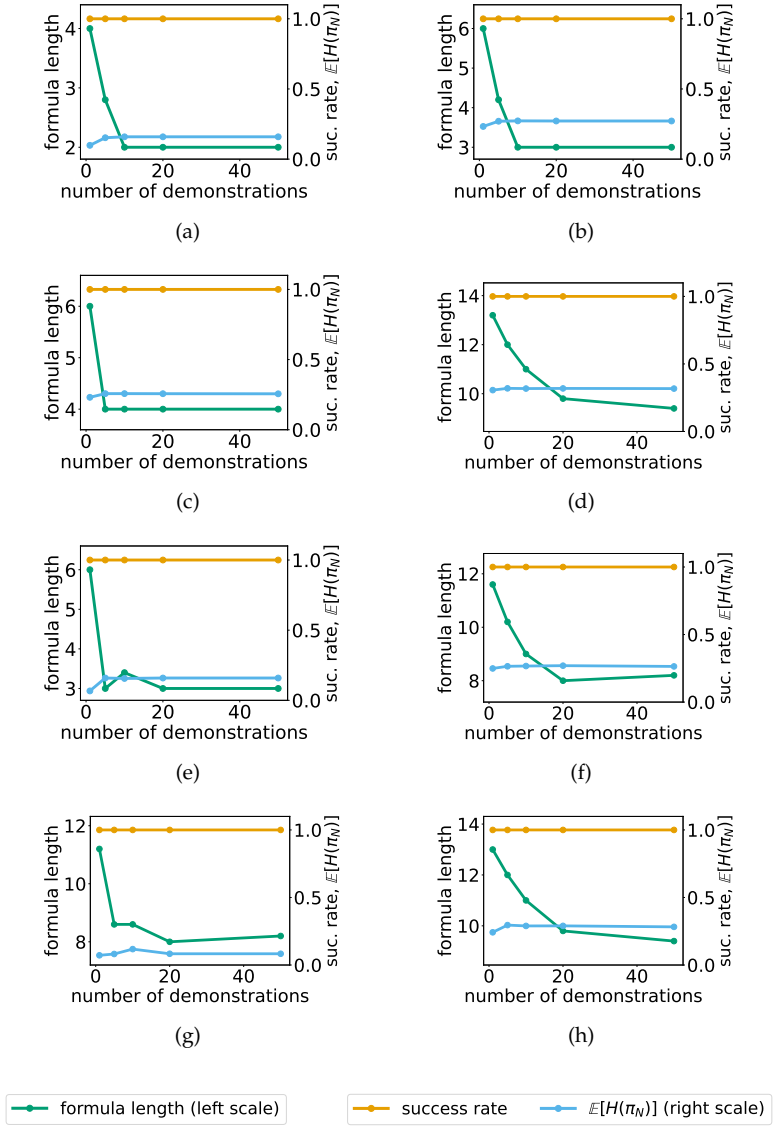


Figure 5.7: Ablation study on the number of expert demonstrations for the Office environments. (a), (b), (c) and (d) correspond with the case AP is given, (e), (f), (g) and (h) show the results for AP unknown.

5.5.3 Ablation Study

In this section, we investigate the effect of the number of expert demonstrations on the formula length, the success rate and the expected entropy of the nominal policy. Results for the Office environment are depicted in figure 5.7. The first two rows shows the results for AP given, the third and fourth row for AP unknown. We can conclude that the success rate is independent of the number of demonstrations. Further, we see the formula length increases for decreasing number of demonstrations while the entropy decreases although in a much smaller rate. This indicates that our method tends to overfit on small number of demonstrations. In general this effect is more pronounced for the case AP is not given and for complex tasks (e.g., Office-4). We attribute this to the larger search space which also facilitates overfitting more heavily.

5.6 Conclusion

We have introduced a novel approach for acquiring task specifications in LTL from expert trajectories. Unlike the majority of existing methods, our method does not require a predefined set of atomic propositions, as it is able to infer formulas and propositions directly from trajectories within the original state space. We explicitly account for the goal-directed nature of trajectories by alternating between the proposed LTL inference and policy update steps. This consideration ensures that only essential components are incorporated into the task specification, thereby leading to the generation of more interpretable formulas and mitigating the risk of overfitting on the provided demonstrations. In this study, we only considered discrete environments. Adapting our methodology to continuous state spaces demands several adjustments. First, the policy update necessitates an approximate method, such as DQN [19], to effectively generalize across states. Additionally, the complexity of the hypothesis space for atomic propositions increases significantly in continuous state spaces, rendering the proposed method for learning atomic propositions unsuitable for such scenarios.

Contribution to explainable artificial intelligence (XAI): Due to the readability of LTL and the clear meaning of each proposition, the learned formulas are interpretable by humans. Additionally, the learned formula provides an explanation of the expert's behavior and can be used to train a RL agent. Unlike most learning-from-demonstration methods, our approach offers an interpretable representation of the RL objective. This interpretability enhances the trustworthiness of our learning agent, as an expert can verify the objectives extracted from demonstrations.

References

[1] A. Pnueli. *The Temporal Logic of Programs*. In 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977, pages 46–57. IEEE Computer Society, 1977. Available from: https://doi.org/10.1109/SFCS.1977.32, doi:10.1109/SFCS.1977.32.

[2] X. Li, C.-I. Vasile, and C. Belta. *Reinforcement learning with temporal logic rewards*. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 3834–3839. IEEE, 2017.

[3] M. L. Littman, U. Topcu, J. Fu, C. Isbell, M. Wen, and J. MacGlashan. *Environment-independent task specifications via GLTL*. arXiv preprint arXiv:1704.04341, 2017.

[4] Z. Xu and U. Topcu. *Transfer of temporal logic formulas in reinforcement learning*. In IJCAI: proceedings of the conference, volume 28, page 4010. NIH Public Access, 2019.

[5] E. Ghiorzi, M. Colledanchise, G. Piquet, S. Bernagozzi, A. Tacchella, and L. Natale. *Learning Linear Temporal Properties for Autonomous Robotic Systems*. IEEE Robotics and Automation Letters, 8(5):2930–2937, 2023.

[6] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. *Concrete problems in AI safety*. arXiv preprint arXiv:1606.06565, 2016.

[7] J. Andreas, D. Klein, and S. Levine. *Modular multitask reinforcement learning with policy sketches*. In International conference on machine learning, pages 166–175. PMLR, 2017.

[8] S. Jha, A. Tiwari, S. A. Seshia, T. Sahai, and N. Shankar. *TeLEx: learning signal temporal logic from positive examples using tightness metric*. Formal Methods in System Design, 54:364–387, 2019.

[9] A. Shah, P. Kamath, J. A. Shah, and S. Li. *Bayesian inference of temporal task specifications from demonstrations*. Advances in Neural Information Processing Systems, 31, 2018.

[10] R. Roy, J.-R. Gaglione, N. Baharisangari, D. Neider, Z. Xu, and U. Topcu. *Learning interpretable temporal properties from positive examples only*. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 37, pages 6507–6515, 2023.

[11] R. Bellman. *A Markovian decision process*. Journal of mathematics and mechanics, pages 679–684, 1957.

[12] P. Abbeel and A. Y. Ng. *Apprenticeship learning via inverse reinforcement learning*. In Proceedings of the twenty-first international conference on Machine learning, page 1, 2004.

[13] J. Ho and S. Ermon. *Generative adversarial imitation learning*. Advances in neural information processing systems, 29, 2016.

[14] M. Baert, P. Mazzaglia, S. Leroux, and P. Simoens. *Maximum Causal Entropy Inverse Constrained Reinforcement Learning*. arXiv preprint arXiv:2305.02857, 2023.

[15] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta. *Q-learning for robust satisfaction of signal temporal logic specifications*. In 2016 IEEE 55th Conference on Decision and Control (CDC), pages 6565–6570. IEEE, 2016.

[16] Z. Xiong, J. Eappen, A. H. Qureshi, and S. Jagannathan. *Constrained Hierarchical Deep Reinforcement Learning with Differentiable Formal Specifications*. 2022.

[17] Y.-L. Kuo, B. Katz, and A. Barbu. *Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of LTL formulas*. In 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 5604–5610. IEEE, 2020.

[18] C. Voloshin, H. Le, S. Chaudhuri, and Y. Yue. *Policy optimization with linear temporal logic constraints*. Advances in Neural Information Processing Systems, 35:17690–17702, 2022.

[19] R. T. Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith. *Reward machines: Exploiting reward function structure in reinforcement learning*. Journal of Artificial Intelligence Research, 73:173–208, 2022.

[20] G. Bombara and C. Belta. *Offline and online learning of signal temporal logic formulae using decision trees*. ACM Transactions on Cyber-Physical Systems, 5(3):1–23, 2021.

[21] Z. Kong, A. Jones, and C. Belta. *Temporal logics for learning and detection of anomalous behavior*. IEEE Transactions on Automatic Control, 62(3):1210–1222, 2016.

[22] R. Yan and A. Julius. *Neural network for weighted signal temporal logic*. arXiv preprint arXiv:2104.05435, 2021.

[23] N. Fronda and H. Abbas. *Differentiable Inference of Temporal Logic Formulas*. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 41(11):4193–4204, 2022.

[24] K. Leung, N. Aréchiga, and M. Pavone. *Backpropagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods*. The International Journal of Robotics Research, 42(6):356–370, 2023.

[25] D. Li, M. Cai, C.-I. Vasile, and R. Tron. *Learning signal temporal logic through neural network for interpretable classification*. In 2023 American Control Conference (ACC), pages 1907–1914. IEEE, 2023.

[26] R. Yan, T. Ma, A. Fokoue, M. Chang, and A. Julius. *Neuro-symbolic Models for Interpretable Time Series Classification using Temporal Logic Description*. In 2022 IEEE International Conference on Data Mining (ICDM), pages 618–627. IEEE, 2022.

[27] M. Vazquez-Chanlatte, S. Jha, A. Tiwari, M. K. Ho, and S. Seshia. *Learning task specifications from demonstrations*. Advances in neural information processing systems, 31, 2018.

[28] T.-Y. Chiu, J. Le Ny, and J.-P. David. *Temporal logic explanations for dynamic decision systems using anchors and Monte Carlo Tree Search*. Artificial Intelligence, 318:103897, 2023.

[29] G. Chou, N. Ozay, and D. Berenson. *Learning temporal logic formulas from suboptimal demonstrations: theory and experiments*. Autonomous Robots, 46(1):149–174, 2022.

[30] A. Camacho, J. Varley, D. Jain, A. Iscen, and D. Kalashnikov. *Disentangled planning and control in vision based robotics via reward machines*. arXiv preprint arXiv:2012.14464, 2020.

[31] Z. Xu, I. Gavran, Y. Ahmad, R. Majumdar, D. Neider, U. Topcu, and B. Wu. *Joint inference of reward machines and policies for reinforcement learning*. In Proceedings of the International Conference on Automated Planning and Scheduling, volume 30, pages 590–598, 2020.

[32] D. Furelos-Blanco, M. Law, A. Jonsson, K. Broda, and A. Russo. *Hierarchies of reward machines*. In International Conference on Machine Learning, pages 10494–10541. PMLR, 2023.

[33] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. *Patterns in property specifications for finite-state verification*. In Proceedings of the 21st international conference on Software engineering, pages 411–420, 1999.

[34] A. Camacho, R. T. Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith. *LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning*. In IJCAI, volume 19, pages 6065–6073, 2019.

[35] A. Duret-Lutz and D. Poitrenaud. *SPOT: An Extensible Model Checking Library Using Transition-Based Generalized Büchi Automata*. In D. DeGroot, P. G. Harrison, H. A. G. Wijshoff, and Z. Segall, editors, 12th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2004), 4-8 October 2004, Vollendam, The Netherlands, pages 76–83. IEEE Computer Society, 2004. Available from: https://doi.org/10.1109/MASCOT.2004.1348184, doi:10.1109/MASCOT.2004.1348184.

[36] J. R. Buchi. *On a decision method in restricted second order arithmetic*. Proc. Internat. Congr. on Logic, Methodology and Philosophy of Science, 1960, 1960.

[37] D. Kasenberg and M. Scheutz. *Interpretable apprenticeship learning with temporal logic specifications*. In 2017 IEEE 56th Annual Conference on Decision and Control (CDC), pages 4914–4921. IEEE, 2017.

6

Learning Task Specifications from Demonstrations as Probabilistic Automata

"Never fight a man with a perm."

IDLES, "Never Fight A Man With A Perm"

Learning Task Specifications from Demonstrations as Probabilistic Automata

M. Baert • S. Leroux • P. Simoens.
Presented at the IEEE conference on robotics and automation (ICRA), 2025.

The method presented in the previous chapter is limited to environments with discrete state spaces. In this chapter, we broaden the scope to address complex environments with continuous state spaces by exploring how sub-goals and their temporal ordering can be extracted from unstructured demonstrations, such as raw video data. We introduce a computationally efficient approach for learning probabilistic deterministic finite automata (PDFA) that capture task structures and expert preferences directly from unstructured demonstrations, such as raw video data. Our approach is based on the assumption that sub-goals are visited more frequently in demonstrations compared to other states. To identify these sub-goals, we apply clustering techniques to detect high-density regions within the state space. We demonstrate this methodology on real-world robotic object manipulation tasks, highlighting its practical applicability.

6.1 Introduction

Robots have become indispensable in manufacturing assembly tasks, yet they often operate under rigidly fixed sequences of sub-tasks. This rigidity causes inefficiencies, such as robots idling when a tool or sub-assembly is unavailable, despite other tasks being ready for execution. For example, in a scenario where two sub-assemblies must be constructed separately and then combined into a final product, a robot constrained by a fixed task sequence would be forced to wait if materials for the first sub-assembly are unavailable, even though it could begin work on the second sub-assembly. Encoding these temporal dependencies into a task specification presents significant challenges for domain experts. Additionally, the variability in execution sequences by different operators renders traditional Learning from Demonstration (LfD) techniques inadequate [1].

To address these challenges, we propose a method for identifying sub-goals and their temporal dependencies from demonstrations. Specifically, we present an algorithm to construct a Probabilistic Deterministic Finite Automaton (PDFA) that models both the task structure and the demonstrators' preferences. In this model, state transitions represent the completion of sub-goals, while states denote the sub-goals that have already been completed. Each transition is assigned a probability reflecting the demonstrators' pref-

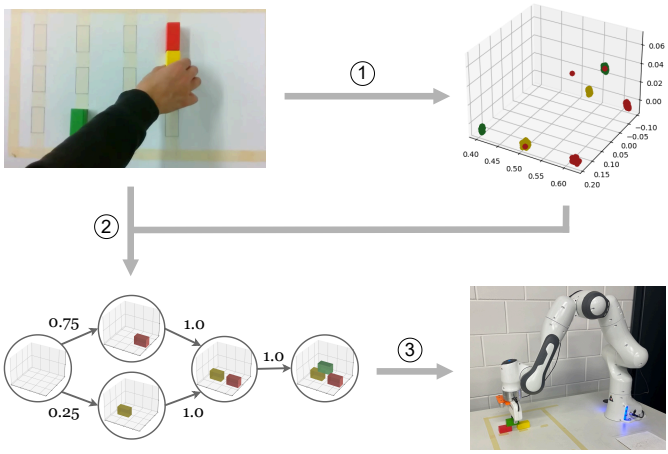


Figure 6.1: 1) Given a set of demonstrations, potential sub-goals are extracted through clustering. 2) From the set of sub-goals and the demonstrations a PDFA is constructed representing task structure and demonstrator preferences. 3) The learned PDFA can be used for task planning.

erences. This PDFA can then be used to plan and replicate expert behaviors. Using a PDFA as a task representation offers several advantages. First, it provides an interpretable model that domain experts can easily validate and modify. Second, PDFAs facilitate online planning, enabling robots to adapt their behavior based on changing conditions. Last, our method accommodates diversity in demonstrations, enhancing the flexibility and robustness of autonomous robots.

Consider the task of building a tower with three colored blocks, where the green block has to be placed on top and the red and yellow blocks form the base (as shown in Figure 6.1). This task can be completed in various ways; for instance, some individuals may start by placing the red block first, while others may begin with the yellow block. To learn a specification for this task, we start by identifying a set of sub-goals, which, in this case, involve placing each block in its correct position. We hypothesize that genuine sub-goals correspond to partial states frequently observed in expert demonstrations, as supported by prior research [2]. To identify these sub-goals, we apply clustering techniques to group similar states and identify the most representative ones. We then construct a PDFA that captures the temporal dependencies between these sub-goals. The variability observed in the demonstrations is reflected in the PDFA's structure, allowing the robot to choose which block to place first while ensuring the task always

concludes with the green block on top. The transition probabilities in the PDFA reflect the preference of the demonstrator, indicating preference of placing the red block before the yellow block. The inferred PDFA can then be used by an autonomous agent, such as a robot arm, to execute the learned task.

Our method addresses the challenge of learning compositional long-horizon tasks, a capability often beyond traditional LfD approaches [1, 3, 4]. Temporal logics provide a framework for representing such complex task specifications. However, existing methods for inferring temporal logic formulas from demonstrations generally require extensive knowledge of the robot's dynamics [5, 6], are limited to simpler tasks [7], or necessitate manual specification of sub-goals [8, 9]. While temporal logics are generally translated into automata for task planning [10], our approach directly infers an automaton. Traditional methods for Deterministic Finite Automaton (DFA) inference can be computationally intensive [11, 12, 13]. By assuming that the extracted sub-goals represent the set of valid transitions, we improve both efficiency and scalability of inferring DFA task specifications.

The main contribution of this chapter is the development of a method for learning sub-goals and their temporal dependencies from demonstrations. This involves the construction of a PDFA that models task structure and demonstrator preferences. This approach accommodates variability in task execution, enhancing flexibility and robustness in robot task planning. Additionally, by introducing an inductive bias based on sub-goals, the method improves the efficiency and scalability of automaton inference for LfD. We demonstrate the practical application of this approach on a variety of object manipulation tasks, showcasing its effectiveness in real-world scenarios.

6.2 Related Work

Many LfD studies focus on learning a policy that maps agent states to actions using techniques such as Reinforcement Learning (RL) [3, 14] or movement primitives [4, 15]. Another set of LfD methods frames the problem as acquiring a standard Markovian reward function [1, 16, 17], which can then be used to learn a policy. However, extracting task specifications from a learned reward function or policy is non-trivial [18], and these methods are generally limited to tasks of limited duration.

To broaden the scope of LfD and tackle more complex tasks, significant efforts have been made to explore more expressive models, such as temporal logics [19], for defining task specifications. Temporal logics have been used for goal definitions in domain-independent planning [20], synthesis of

reactive systems [21], and RL [22, 23, 24]. However, writing correct formal specifications requires domain knowledge. Recent studies have proposed learning Linear Temporal Logic (LTL) formulas from data, often relying on labeled datasets of positive and negative traces [25, 26, 27]. In the context of LfD, collecting negative examples can be challenging and dangerous. Recent methods have focused on learning LTL exclusively from positive examples [8, 9, 28]. However, these approaches can be slow due to the extensive formula exploration space or limit their search to a restricted set of formula templates. Additionally, these methods assume a world state description in terms of Boolean propositions for each time step in the demonstrations. Studies like [7] and [29] allow learning Signal Temporal Logic (STL) formulas from continuous, unstructured demonstrations but are limited to very simple formulas. The method of [5] can learn both the formula structure and the set of propositions mapped to the world state but requires complete knowledge of the environment's dynamics. In contrast, our method only makes certain assumptions about the task structure and underlying sub-goals. To use temporal logic formulas for planning, they typically need to be translated to automata, which can lead to the state-explosion problem [10]. One way to overcome this is by directly learning a DFA. [30] proposes a method for learning transitions between states given the DFA structure. [13] extends DFA inference based on state merging to learn PDFAs under safety constraints. However, the merging process can be computationally expensive and may lead to exponential growth in complexity with the number of states.

Similar to sub-goal inference in our method, many methods for learning plan specifications from demonstrations first segment each demonstration into sub-tasks. Previous work relies on manual segmentation [31], hidden Markov models [32, 33], Chinese restaurant processes [34], or seq-2-seq models [35]. [36] uses support vector machines and Gaussian mixture models to assign each time step to a motion primitive from a movement library, while [37] uses narrations of the human demonstrator to identify boundaries between primitives. Unlike our method, these techniques require extensive hyperparameter tuning, consider only sequential task structures, or demand additional efforts from the demonstrator.

There is extensive research on inferring DFAs from examples, which extends beyond the scope of LfD. As demonstrations typically consist only of valid trajectories, we focus on methods for learning DFAs from positive examples. Traditional DFA inference methods include state-merging algorithms that iteratively generalize a DFA by merging states [38, 11], incremental approaches that refine a set of candidate automata as new examples are

introduced [39], and genetic programming techniques that evolve DFAs over time [40, 12]. These methods are versatile, making few assumptions about the underlying system generating the traces, and thus are applicable across various domains. However, this generality often comes at the cost of significant computational resources and time. Our approach aims to improve the efficiency and scalability of DFA inference in the context of LfD by introducing specific assumptions about the task structure and sub-goals.

6.3 Problem Formulation

6.3.1 States, Demonstrations and Sub-Goals

Let S be the set of world states. Each world state $s \in S$ is represented as a vector $s = (f_0, f_1, ..., f_{n-1})$, where $f_i \in \mathcal{F}$ are features describing the state, and $\mathcal{F}$ denotes the set of all features. These features could include positions, object properties, agent statuses, environmental conditions, etc. A demonstration $d = \{s_0, s_1, ..., s_{T-1}\}$ is defined as a series of states with T denoting the duration of the demonstration measured in discrete time steps. A sub-goal $g = (c, r)$ is defined by a center point c using a subset of features $\mathcal{F}_g \subseteq \mathcal{F}$ and a radius r, forming a $\mid \mathcal{F}_g \mid$-dimensional ball. We define a mapping function $\phi(\mathcal{F}_g, s)$ which extracts, from a state s, a partial state $\hat{s}$ consisting of the features specified by $\mathcal{F}_g$. To determine if a state s satisfies a sub-goal, we check if the Euclidean distance between the partial state $\hat{s} = \phi(\mathcal{F}_g, s)$ and the center is smaller or equal to the specified radius r.

6.3.2 Task Specification

We assume that the human demonstrator has a task in mind that can be completed in finite time. Our objective is to learn this task in the form of a DFA, which captures the valid sequences of sub-goals as demonstrated by the experts.

Definition 10 *(DFA)* A Deterministic Finite Automaton (DFA) is a tuple $\mathcal{A} = (Q, q_0, \Sigma, \delta, F)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, Σ is the input alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function and $F \subseteq Q$ is the set of accepting states.

A word ω is composed of a finite sequence of symbols $\sigma \in \Sigma$: $\omega = \sigma_0\sigma_1...\sigma_{n-1}$. We define ϵ as the empty word. The concatenation of a symbol σ to a word ω is defined as $\omega' = \omega \cdot \sigma$. A word ω generates a trajectory of DFA states $\tau = q_0q_1...q_n$ if $q_n = \delta(q_{n-1}, \sigma_{n-1})$ for all $n \geq 0$. A finite input word ω is accepted by the automaton if the corresponding generated trajectory ends in a state within the set of accepting states: $q_n \in F$. The (accepted) language of $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$, is the set of all accepted input words. In addition to learning the DFA, we aim to encode the demonstra-

tor's preferences by quantifying the probabilities of transitions between DFA states. Thus, our goal is to learn a PDFA that captures both the valid sequences (accepting words) and their corresponding probabilities.

Definition 11 *(PDFA)* A probabilistic DFA (PDFA) is a tuple $\mathcal{A}^{\mathbb{P}} = (\mathcal{A}, \delta_{\mathbb{P}}, F_{\mathbb{P}})$, where $\mathcal{A}$ is a DFA, $\delta_{\mathbb{P}} : \delta \to [0,1]$ assigns a probability to each transition in δ, ensuring that $\sum_{\sigma \in \Sigma} \delta_{\mathbb{P}}(q, \sigma, \delta(q, \sigma)) = 1$ for every $q \in Q$. Additionally, $F_{\mathbb{P}} : Q \to [0,1]$ assigns a probability of terminating at each state, with $F_{\mathbb{P}}(q) = 0$ if $q \notin F$.

Consider a word $\omega = \sigma_0 \sigma_1 ... \sigma_{n-1}$ and its induced trace $\tau = q_0 q_1 ... q_n$ on PDFA $\mathcal{A}^{\mathbb{P}}$. The probability of ω is given by

$$P(\omega) = \prod_{i=0}^{n-1} \delta_{\mathbb{P}}(q_i, \sigma_i, q_{i+1}) \cdot F_{\mathbb{P}}(q_n). \tag{6.1}$$

We say $\mathcal{A}^{\mathbb{P}}$ accepts ω iff $P(\omega) > 0$. The language of $\mathcal{A}^{\mathbb{P}}$ is the set of words with non-zero probabilities.

6.3.3 Problem

Given a set of demonstrations $D = \{d_0, d_1, ..., d_{m-1}\}$ we want to (1) learn a set of sub-goals G representing the different steps of the demonstrated task; (2) learn a PDFA $\mathcal{A}^{\mathbb{P}}$ which represents the valid temporal orderings of sub-goals and the expert's preferences; (3) use the learned PDFA to compute a plan for an artificial agent to replicate the most preferred expert behavior. We consider frequently observed partial states as sub-goals and use clustering to infer a set of sub-goals from demonstrations as detailed in Section 6.4.1. In Section 6.4.2, we describe how a PDFA can be constructed. This PDFA can be used for planning as described in Section 6.4.3.

6.4 Methodology

6.4.1 Sub-Goal Inference

Each sub-goal is defined by a subset of all features. We denote the candidate feature subsets as $\mathcal{C}(\mathcal{F})$, representing a collection of subsets of F. Each sub-goal is thus associated with one of these subsets: $\mathcal{F}_g \in \mathcal{C}(\mathcal{F})$ for all $g \in G$. We have access to a set of demonstrations $D = d_0, d_1, \ldots, d_{m-1}$. For each subset $\mathcal{F}_i \in \mathcal{C}(\mathcal{F})$, we construct a corresponding dataset by ensuring that, for each demonstration $d \in D$, each state $s \in d$ is transformed to a partial state $\hat{s} = \phi(\mathcal{F}_i, s)$ such that the partial state vector contains values only for the features $f \in \mathcal{F}_i$. This process results in a new collection of datasets $\hat{D} = \hat{D}_0, \hat{D}1, \ldots, \hat{D}_{C-1}$, where C denotes the number of subsets in $\mathcal{C}(\mathcal{F})$. We reason that genuine sub-goals should be frequently observed in the expert

demonstrations. To identify these regions, we use DBSCAN clustering [41] on all datasets in $\hat{D}$. DBSCAN is robust to noise from incorrect object detections and does not require prior knowledge of the number of clusters, which is essential given that the number of sub-goals is unknown. After clustering, we eliminate clusters that intersect with initial states (i.e., states at the beginning of a demonstration). The remaining clusters form the set of sub-goals G. Note that each cluster, like a sub-goal, is defined by a point in a subspace of $\mathcal{F}$ and a radius.

6.4.2 PDFA Inference

First, we construct an alphabet Σ such that each symbol $\sigma \in \Sigma$ represents a single sub-goal with σ_g denoting the symbol corresponding to sub-goal g. Next, the set of demonstrations D is converted into a set of words Ω, such that each demonstration is represented by a sequence of sub-goals. The conversion process, detailed in Algorithm 6.1, initializes an empty word $\omega = \epsilon$. At each time step, we verify if any sub-goals are completed. Upon completion, the corresponding sub-goal symbol σ_g is appended to ω. We assume that the preference for a particular word is correlated with its frequency of occurrence in Ω. Therefore, preferences can be quantified as weights assigned to traces and normalized across the entire language, forming a probability distribution over $\mathcal{L}(\mathcal{A})$. A higher probability of an accepting word indicates greater preference by the demonstrator. Specifically, the demonstrator prefers ω over ω' if and only if $P(\omega) > P(\omega')$.

Algorithm 6.1: Trajectory to word conversion

Input: sub-goal radius r, alphabet Σ, demonstration d

```
ω ← ϵ
for s ∈ d do
    for g ∈ G do
        if σ_g ∉ ω then
            c, r ← g
            if ||φ(F_g, s) − c||_2 ≤ r then
                ω ← ω · σ_g
                break
            end
        end
    end
end
return ω
```

Algorithm 6.2: DFA Inference

Input: alphabet Σ, set of words Ω representing expert demonstrations

$q_0 \leftarrow$ init_new_state()
$Q \leftarrow \{q_0\}$
$F \leftarrow \emptyset$
$\delta(q, \sigma) \leftarrow$ undefined $\quad \forall q \in Q, \forall \sigma \in \Sigma$
$V \leftarrow \mathbf{0}_{2^{|\Sigma|} \times 2^{|\Sigma|}}$
$\psi(A) \leftarrow$ undefined $\quad \forall A \subseteq \Sigma$
for $\omega \in \Omega$ **do**
  $\Lambda \leftarrow \emptyset$
  $q \leftarrow q_0$
  for $\sigma \in \omega$ **do**
    $\Lambda \leftarrow \Lambda \cup \{\sigma\}$
    if $\psi(\Lambda) \neq$ *undefined* **then**
      if $\delta(q, \sigma) =$ *undefined* **then**

$$\delta(\hat{q}, \hat{\sigma}) \leftarrow \begin{cases} \psi(\Lambda) & \text{if } \hat{q} = q \text{ and } \hat{\sigma} = \sigma \\ \delta(\hat{q}, \hat{\sigma}) & \text{otherwise} \end{cases}$$

      end
    else
      $q_{\text{new}} \leftarrow$ init_new_state()
      $Q \leftarrow Q \cup \{q_{\text{new}}\}$

$$\psi(A) \leftarrow \begin{cases} q_{\text{new}} & \text{if } A = \Lambda \\ \psi(A) & \text{otherwise} \end{cases}$$

$$\delta(\hat{q}, \hat{\sigma}) \leftarrow \begin{cases} q_{\text{new}} & \text{if } \hat{q} = q \text{ and } \hat{\sigma} = \sigma \\ \text{undefined} & \text{if } \hat{q} = q_{\text{new}} \\ \delta(\hat{q}, \hat{\sigma}) & \text{otherwise} \end{cases}$$

    end
    $q' \leftarrow \delta(q, \sigma)$
    $V[q][q'] \leftarrow V[q][q'] + 1$
    $q \leftarrow q'$
  end
  if $q \notin F$ **then**
    $F \leftarrow F \cup \{q\}$
  end
end
Return $(Q, q_0, \Sigma, \delta, F)$, V

Algorithm 6.2 describes the process for inferring a DFA that captures the sub-goal ordering observed in the demonstrations. Additionally, a frequency map V is constructed to record the transition frequencies between each pair of DFA states. First, the algorithm initializes a DFA $\mathcal{A}$ with the initial state q_0 and the alphabet Σ representing the extracted sub-goals. The set of accepting states is initialized as an empty set $F = \emptyset$, and the transition function δ is initialized such that there are no valid transitions from the initial state q_0. All values in the frequency map V are initialized to zero. A mapping function $\psi(A) : 2^{\Sigma} \rightarrow Q$ is defined, which maps a subset of symbols to a DFA state, where 2^{Σ} denotes the power set of the alphabet Σ. This mapping ψ is initially set to assign no subsets of Σ to any states in Q.

The algorithm then iterates over all words $\omega \in \Omega$. For each word, the current state q is initialized to q_0. An empty set Λ is initialized to represent the set of already completed sub-goals. For each symbol σ in the word ω, σ is added to Λ. This process ensures that Λ correctly represents the set of sub-goals completed so far. Each word ω generates a trajectory that ends in a specific DFA state q. When there are multiple valid sub-goal orderings, the algorithm ensures that all corresponding words generate trajectories ending in the same state, maintaining a direct relation between DFA states and completed sub-goals. If for the current state q and symbol σ, $\psi(\Lambda) \neq$ undefined and $\delta(q, \sigma) =$ undefined, it indicates that the same set of sub-goals has been completed in a different order. In such cases, only the transition function δ needs to be updated. If the current set of completed sub-goals Λ has not been observed before, a new state q_{new} should be added. The transition function δ is updated, and ψ is modified to map Λ to q_{new}. This algorithm assures that all demonstrations are represented by the DFA's language $\mathcal{L}(\mathcal{A})$.

Each symbol σ induces a transition from state q to q', for each transition, the corresponding entry in V is updated. The state reached at the end of each word $\omega \in \Omega$ is added to the set of accepting states F. If different demonstrations consist of different sets of completed sub-goals, F will contain multiple accepting states.

The probabilistic transition function $\delta_{\mathbb{P}}$ and the set of probabilistic accepting states $F_{\mathbb{P}}$ can be easily extracted from V, to construct a PDFA:

$$\delta_{\mathbb{P}}(q,\sigma,q') = \begin{cases} \dfrac{V[q][q']}{\sum_{q''\in Q} V[q][q'']} & \text{if } \delta(q,\sigma) = q' \\ 0 & \text{otherwise,} \end{cases} \tag{6.2}$$

$$F_{\mathbb{P}}(q) = \begin{cases} \dfrac{\sum_{q'\in Q} V[q'][q]}{\sum_{q'\in Q}\sum_{q''\in F} V[q'][q'']} & \text{if } q \in F \\ 0 & \text{if } q \notin F. \end{cases} \tag{6.3}$$

6.4.3 Planning With PDFAs

Once a task is represented as a PDFA, our goal is to generate a plan that not only accomplishes the task but also aligns with the demonstrator's preferences. We assume the existence of a low-level controller, such as a model-based controller or a trained RL policy, capable of achieving individual sub-goals. Each transition in the PDFA corresponds to a single execution of this controller, achieving the associated sub-goal. Consequently, replicating the expert's behavior is simplified to a process of greedily selecting the transition with the highest probability at each PDFA state, continuing this process until an accepting state is reached.

6.5 Experiments

We extensively evaluated our method on object manipulation tasks using unstructured human demonstrations. We deployed our approach on a physical robot to replicate expert behaviors and tested it in two simulated environments to demonstrate its versatility. The first simulated environment involved a surveillance task executed by a quadcopter using a path and motion planning algorithm for low-level control. The second involved using RL for low-level control with a two-jointed robot arm.

6.5.1 Object Manipulation

We designed a series of multistep object manipulation tasks using wooden blocks to evaluate our method. We utilized a Franka Emika Panda robot, with demonstrations performed by a human expert and recorded with an RGB-D camera mounted on the robot's end effector (the right picture in Figure 6.1 depicts our setup). The end effector was positioned to observe the workspace from a predefined location during demonstrations. We used a thresholded depth image to detect the location and pose of objects, which, despite its simplicity, was effective for the simple shapes involved. RGB color detection was used to distinguish and track objects. The world state

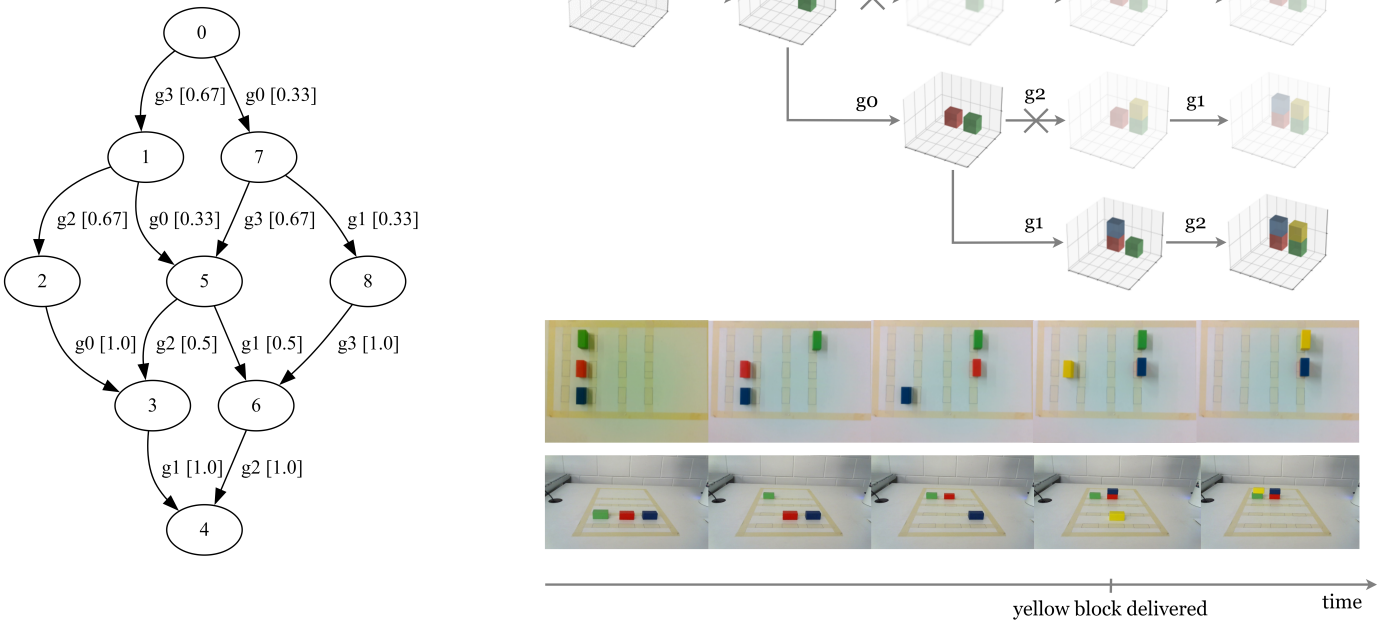


Figure 6.2: **Left**: Inferred PDFA (from 9 demonstrations) for a task involving the construction of two stacks of two blocks, with each block required to be placed in a designated location. **Top right**: The initial plan selects sub-goals based on the highest transition probabilities. However, during execution, the unavailability of the yellow block prevents the agent from completing sub-goal 2, necessitating two re-planning steps. **Bottom right**: Snapshots of the environment during task execution. The top row displays frames captured by the camera mounted on the robot's end effector, while the bottom row shows side-view frames captured by an additional camera.

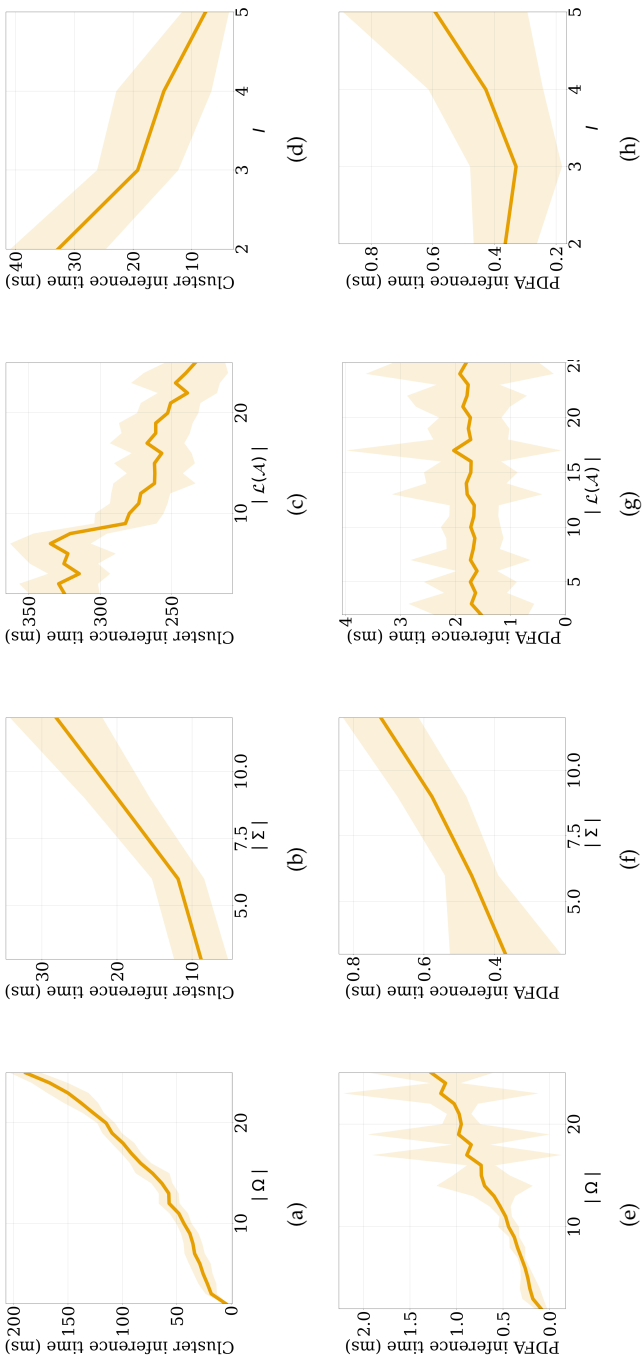


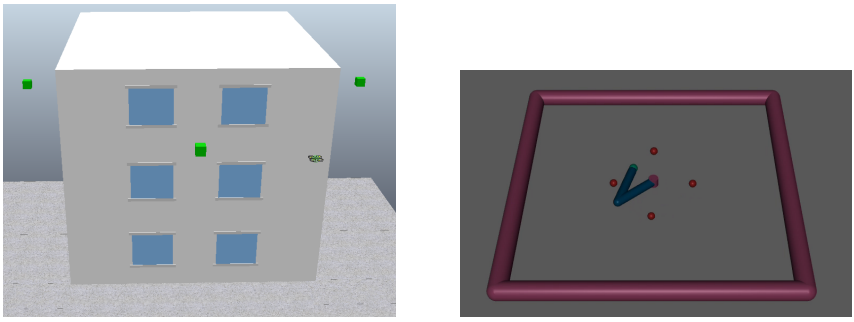
Figure 6.3: Effect of different task properties on the cluster and PDFA inference time. The investigated properties are the number of demonstrations $|\Omega|$ (Figure 6.3(a) and 6.3(e)), the number of sub-goals $|\Sigma|$ (Figure 6.3(b) and 6.3(f)), the language size $\mathcal{L}(\mathcal{A})$ (Figure 6.3(c) and 6.3(g)) and the number of objects l (Figure 6.3(d) and 6.3(h)).

vector consists of the x-, y-, and z-coordinates of each object. For each object, a subset of features is added to the set of candidate subspaces $\mathcal{C}(\mathcal{F})$ that define its coordinates.

For deployment, the robot's end effector was moved to the observation location to update the world state based on the current object positions. If an object was not detected, the corresponding features were set as undefined. After selecting a sub-goal, we used a low-level Cartesian impedance controller from the SERL Franka library [42] to complete the action. Post-action, the end effector returned to its initial position for another world state update. Because each sub-goal is defined as a location of a single object, we can assess the reachability of a sub-goal by checking if the corresponding object is detected.

Figure 6.2 illustrates a learned PDFA for a task involving stacking two sets of blocks. The automaton indicates six valid orderings of the sub-goals, with the preferred sequence being g3, g2, g0, g1. The right side shows this initial plan in the top row. However, when it turns out that the next sub-goal is unreachable (in this case g2 because the yellow block is not detected), a new plan is generated (see second and third row).

We investigated the impact of four task properties on clustering and PDFA inference time: the number of demonstrations $\mid \Omega \mid$, the number of sub-goals $\mid \Sigma \mid$, the language size of the extracted PDFA $\mid \mathcal{L}(\mathcal{A}) \mid$, and the number of objects I. Experiments were carefully designed to isolate the influence of each property while keeping others constant. Figure 6.3 shows the results. To investigate the influence of the number of demonstrations $\mid \Omega \mid$, we used a task of placing four blocks at specified locations. For the clustering, inference time increases exponentially with the number of demonstrations. Our automata inference algorithm, however, shows linear time complexity due to the single loop over all demonstrations in Algorithm 6.2. To assess the effect of the number of sub-goals $\mid \Sigma \mid$, we designed tasks involving the stacking and unstacking of three blocks. We conducted four experiments. For $\mid \Sigma \mid = 3$, building a stack of three blocks. For $\mid \Sigma \mid = 6$, first building the same stack and then a second stack using the same blocks but at a different location, with the top and bottom blocks swapped. Similarly, we designed experiments for $\mid \Sigma \mid = 9$ and $\mid \Sigma \mid = 12$. Both clustering and PDFA inference times scale linearly with $\mid \Sigma \mid$. The language $\mathcal{L}(\mathcal{A})$ of an automaton $\mathcal{A}$ represents the variety of ways to perform a task. For this, we re-use the task of moving four blocks to specified locations. The demonstrations contain $4! = 24$ different ways of performing this task. As expected, the language size has no effect on PDFA inference time. Interestingly, clustering time decreases as $\mid \mathcal{L}(\mathcal{A}) \mid$ increases. We also investigated the effect of the



(a) (b)

Figure 6.4: (a) drone surveillance scenario and (b) two-jointed robot arm.

number of objects I. We expected no effect on both clustering and PDFA inference times. However, clustering time decreases with increasing I. This is due to our experiment design: for stacking multiple blocks (up to 5), to keep $\mid \Sigma \mid$ constant, we need to move blocks from the stack when $I < 5$. When $I = 5$, each feature subspace consists of a single sub-goal. For $I < 5$, some objects correspond to two or more sub-goals, making clustering harder and more time-consuming. There is a very small, unexpected increase in PDFA inference time.

We compared our DFA inference procedure with a method using satisfiability (SAT) solvers [43] and another method for inferring LTL specifications from positive examples [9]. Our approach was significantly faster, with SAT solvers timing out on tasks with more than three sub-goals and the LTL method failing to infer correct specifications even for simple tasks. For both methods, we provide demonstrations already converted into words using the proposed sub-goal extraction method and Algorithm 6.1.

6.5.2 Drone Surveillance

In a simulated environment inspired by [5], we tasked a quadcopter with visiting three specified locations using a path and motion planning algorithm for low-level control. For this environment, a state is defined by the drone's coordinates and we define $\mathcal{C}(\mathcal{F}) = \{\mathcal{F}\}$. Our method successfully identified the sub-goals and their order. The agent moved effectively between sub-goals using RTT motion planning and inverse kinematics. Figure 6.4(a) depicts this environment, the green boxes represent the inferred sub-goals.

6.5.3 Two-Jointed Robot Arm

To demonstrate compatibility with RL, we designed a task where a two-jointed robot arm's end effector moves to specific locations in a specified order. This environment is based on the Gymnasium Reacher environment [44], see Figure 6.4(b) for a snapshot. We defined 4 sub-goals, one on the positive side of the x-axis, one on the negative side of the x-axis and similarly two on the y-axis. We defined a single feature subset consisting of the x- and y-coordinates of the end effector. Furthermore, we trained an RL agent on moving its end effector to random coordinates. Our method successfully extracted the four sub-goals and constructed a PDFA based on expert demonstrations, which alternated between clockwise and counter-clockwise sub-goal sequences. Next, we successfully deployed the trained policy to perform the task.

6.6 Conclusions

In this chapter, we presented a novel method for learning PDFAs from unstructured human demonstrations to capture task structure and demonstrator preferences allowing for robust replication of expert behaviors. Through extensive evaluations on object manipulation tasks with a physical robot arm, we demonstrated the effectiveness and efficiency of our method. Additionally, we validated the applicability of our approach in other domains by deploying it in simulated environments. A limitation of our work is that we require domain knowledge to define the candidate feature subsets $\mathcal{C}(\mathcal{F})$ manually. Future work could investigate feature importance methods and dimensionality reduction techniques to automate this step. Instead of using a pre-trained a policy, a policy can be learned for each transition in the PDFA, or one for each sub-goal. This relates to hierarchical RL as each policy can be seen as an individual option [33].

References

[1] J. Ho and S. Ermon. *Generative adversarial imitation learning*. Advances in neural information processing systems, 29, 2016.

[2] B. Ghazanfari, F. Afghah, and M. E. Taylor. *Sequential association rule mining for autonomously extracting hierarchical task structures in reinforcement learning*. IEEE Access, 8:11782–11799, 2020.

[3] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel. *Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation*. In 2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018, pages 1–8. IEEE, 2018. Available from: https://doi.org/10.1109/ICRA.2018.8461249, doi:10.1109/ICRA.2018.8461249.

[4] J. Kober and J. Peters. *Learning motor primitives for robotics*. In 2009 IEEE International Conference on Robotics and Automation, pages 2112–2118. IEEE, 2009.

[5] G. Chou, N. Ozay, and D. Berenson. *Learning temporal logic formulas from suboptimal demonstrations: theory and experiments*. Autonomous Robots, 46(1):149–174, 2022.

[6] W. Liu, D. Li, E. Aasi, R. Tron, and C. Belta. *Interpretable Generative Adversarial Imitation Learning*. CoRR, abs/2402.10310, 2024. Available from: https://doi.org/10.48550/arXiv.2402.10310, arXiv:2402.10310, doi:10.48550/ARXIV.2402.10310.

[7] S. Jha, A. Tiwari, S. A. Seshia, T. Sahai, and N. Shankar. *TeLEx: learning signal temporal logic from positive examples using tightness metric*. Formal Methods in System Design, 54:364–387, 2019.

[8] A. Shah, P. Kamath, J. A. Shah, and S. Li. *Bayesian inference of temporal task specifications from demonstrations*. Advances in Neural Information Processing Systems, 31, 2018.

[9] R. Roy, J.-R. Gaglione, N. Baharisangari, D. Neider, Z. Xu, and U. Topcu. *Learning interpretable temporal properties from positive examples only*. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 37, pages 6507–6515, 2023.

[10] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT press, 2008.

[11] K. J. Lang, B. A. Pearlmutter, and R. A. Price. *Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm*.

In International Colloquium on Grammatical Inference, pages 1–12. Springer, 1998.

[12] G. Petasis, G. Paliouras, C. D. Spyropoulos, and C. Halatsis. *eg-GRIDS: Context-free grammatical inference from positive examples using genetic search*. In Grammatical Inference: Algorithms and Applications: 7th International Colloquium, ICGI 2004, Athens, Greece, October 11-13, 2004. Proceedings 7, pages 223–234. Springer, 2004.

[13] K. Watanabe, N. Renninger, S. Sankaranarayanan, and M. Lahijanian. *Probabilistic specification learning for planning with safety constraints*. In 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 6558–6565. IEEE, 2021.

[14] A. Mandlekar, D. Xu, R. Martín-Martín, S. Savarese, and L. Fei-Fei. *GTI: Learning to Generalize across Long-Horizon Tasks from Human Demonstrations*. In M. Toussaint, A. Bicchi, and T. Hermans, editors, Robotics: Science and Systems XVI, Virtual Event / Corvalis, Oregon, USA, July 12-16, 2020, 2020. Available from: https://doi.org/10.15607/RSS.2020.XVI.061, doi:10.15607/RSS.2020.XVI.061.

[15] S. Schaal. *Dynamic movement primitives-a framework for motor control in humans and humanoid robotics*. In Adaptive motion of animals and machines, pages 261–280. Springer, 2006.

[16] P. Abbeel and A. Y. Ng. *Apprenticeship learning via inverse reinforcement learning*. In Proceedings of the twenty-first international conference on Machine learning, page 1, 2004.

[17] M. Baert, P. Mazzaglia, S. Leroux, and P. Simoens. *Maximum Causal Entropy Inverse Constrained Reinforcement Learning*. arXiv preprint arXiv:2305.02857, 2023.

[18] T. Arnold and D. Kasenberg. *Value Alignment or Misalignment–What Will Keep Systems Accountable?* In AAAI Workshop on AI, Ethics, and Society, 2017.

[19] A. Pnueli. *The temporal logic of programs*. In 18th annual symposium on foundations of computer science (sfcs 1977), pages 46–57. ieee, 1977.

[20] J. Kim, C. Banks, and J. Shah. *Collaborative planning with encoding of users' high-level strategies*. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 31, 2017.

[21] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. *Temporal-logic-based reactive mission and motion planning*. IEEE transactions on robotics, 25(6):1370–1381, 2009.

[22] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta. *Q-learning for robust satisfaction of signal temporal logic specifications*. In 2016 IEEE 55th Conference on Decision and Control (CDC), pages 6565–6570. IEEE, 2016.

[23] X. Li, C.-I. Vasile, and C. Belta. *Reinforcement learning with temporal logic rewards*. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 3834–3839. IEEE, 2017.

[24] Z. Xiong, J. Eappen, A. H. Qureshi, and S. Jagannathan. *Constrained Hierarchical Deep Reinforcement Learning with Differentiable Formal Specifications*. 2022.

[25] G. Bombara and C. Belta. *Offline and online learning of signal temporal logic formulae using decision trees*. ACM Transactions on Cyber-Physical Systems, 5(3):1–23, 2021.

[26] Z. Kong, A. Jones, and C. Belta. *Temporal logics for learning and detection of anomalous behavior*. IEEE Transactions on Automatic Control, 62(3):1210–1222, 2016.

[27] R. Yan, T. Ma, A. Fokoue, M. Chang, and A. Julius. *Neuro-symbolic Models for Interpretable Time Series Classification using Temporal Logic Description*. In 2022 IEEE International Conference on Data Mining (ICDM), pages 618–627. IEEE, 2022.

[28] M. Vazquez-Chanlatte, S. Jha, A. Tiwari, M. K. Ho, and S. Seshia. *Learning task specifications from demonstrations*. Advances in neural information processing systems, 31, 2018.

[29] T.-Y. Chiu, J. Le Ny, and J.-P. David. *Temporal logic explanations for dynamic decision systems using anchors and Monte Carlo Tree Search*. Artificial Intelligence, 318:103897, 2023.

[30] B. Araki, K. Vodrahalli, T. Leech, C.-I. Vasile, M. D. Donahue, and D. L. Rus. *Learning to plan with logical automata*. 2019.

[31] S. Ekvall and D. Kragic. *Robot learning from demonstration: a task-level planning approach*. International Journal of Advanced Robotic Systems, 5(3):33, 2008.

[32] S. Niekum, S. Osentoski, G. Konidaris, S. Chitta, B. Marthi, and A. G. Barto. *Learning grounded finite-state representations from unstructured demonstrations*. The International Journal of Robotics Research, 34(2):131–157, 2015.

[33] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto. *Robot learning from demonstration by constructing skill trees*. The International Journal of Robotics Research, 31(3):360–375, 2012.

[34] D. H. Grollman and O. C. Jenkins. *Incremental learning of subtasks from unsegmented demonstration*. In 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 261–266. IEEE, 2010.

[35] S. Pirk, K. Hausman, A. Toshev, and M. Khansari. *Modeling Long-horizon Tasks as Sequential Interaction Landscapes*. In J. Kober, F. Ramos, and C. J. Tomlin, editors, 4th Conference on Robot Learning, CoRL 2020, 16-18 November 2020, Virtual Event / Cambridge, MA, USA, volume 155 of *Proceedings of Machine Learning Research*, pages 471–484. PMLR, 2020. Available from: https://proceedings.mlr.press/v155/pirk21a.html.

[36] S. Manschitz, J. Kober, M. Gienger, and J. Peters. *Learning to sequence movement primitives from demonstrations*. In 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 4414–4421. IEEE, 2014.

[37] A. Mohseni-Kabir, C. Li, V. Wu, D. Miller, B. Hylak, S. Chernova, D. Berenson, C. Sidner, and C. Rich. *Simultaneous learning of hierarchy and primitives for complex robot tasks*. Autonomous Robots, 43:859–874, 2019.

[38] K. J. Lang. *Random DFA's can be approximately learned from sparse uniform examples*. In Proceedings of the fifth annual workshop on Computational learning theory, pages 45–52, 1992.

[39] R. L. Rivest and R. E. Schapire. *Inference of finite automata using homing sequences*. In Proceedings of the twenty-first annual ACM symposium on Theory of computing, pages 411–420, 1989.

[40] P. Dupont. *Regular grammatical inference from positive and negative samples by genetic search: the GIG method*. In International Colloquium on Grammatical Inference, pages 236–245. Springer, 1994.

[41] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. *A density-based algorithm for discovering clusters in large spatial databases with noise*. In kdd, pages 226–231, 1996.

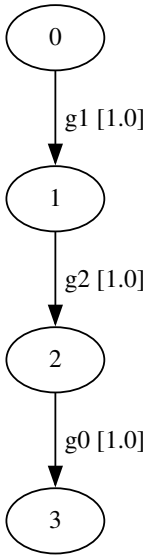
[42] J. Luo, Z. Hu, C. Xu, Y. L. Tan, J. Berg, A. Sharma, S. Schaal, C. Finn, A. Gupta, and S. Levine. *SERL: A Software Suite for Sample-Efficient Robotic Reinforcement Learning*, 2024. arXiv:2401.16013.

[43] M. J. Heule and S. Verwer. *Exact DFA identification using SAT solvers*. In Grammatical Inference: Theoretical Results and Applications: 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings 10, pages 66–79. Springer, 2010.

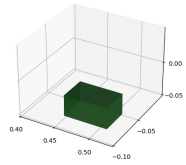
[44] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, et al. *Gymnasium: A Standard Interface for Reinforcement Learning Environments*. arXiv preprint arXiv:2407.17032, 2024.

6.7 Supplementary Data

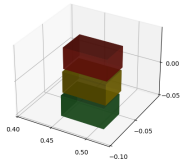
Table 6.1 summarizes all the tested scenarios. This section offers further examples of the inferred PDFAs, along with visualizations of the object placements associated with each PDFA state.



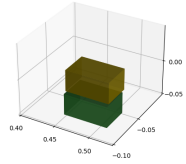
(a) automaton



(b) q_1

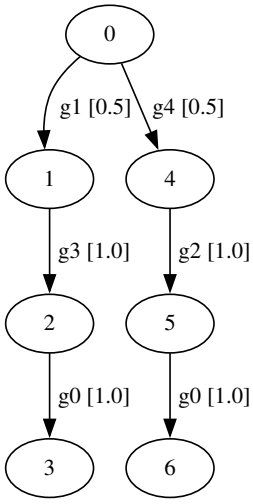


(d) q_3

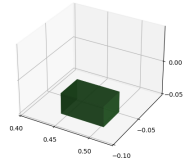


(c) q_2

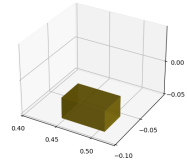
Figure 6.5: Task 0



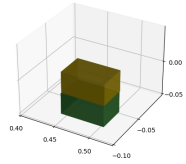
(a) automaton



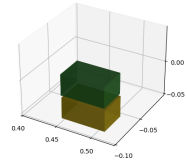
(b) q_1



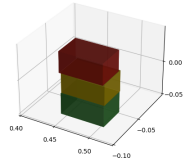
(e) q_4



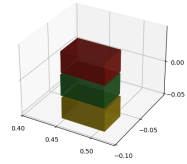
(c) q_2



(f) q_5

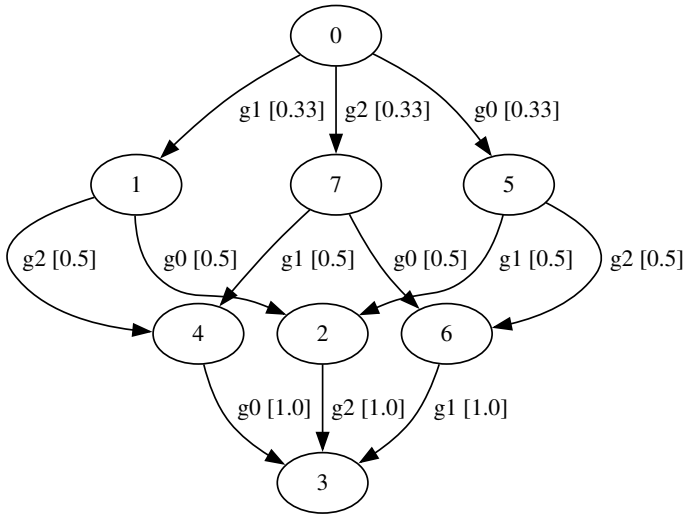


(d) q_3

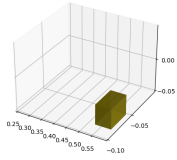


(g) q_6

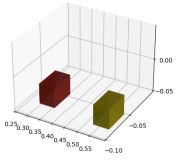
Figure 6.6: Task 1



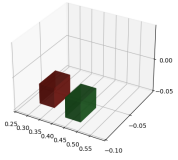
(a) automaton



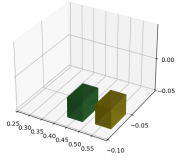
(b) q_1



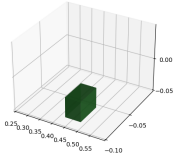
(e) q_4



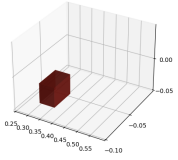
(g) q_6



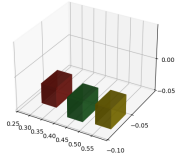
(c) q_2



(f) q_5



(h) q_7



(d) q_3

Figure 6.7: Task 2

Table 6.1: Overview of the different test scenarios.

Task ID	Description
0	stack three blocks in the order: green (bottom), yellow (middle), red (top)
1	stack three blocks such that the red block is at the top
2	move three blocks to a specified location
3	build a tower with the red and yellow blocks as the base, and the green block on top
4	build a tower of 3 blocks (in a specific order) and then move the tower to the marked location while only grasping one block at a time
5	build two towers of 2 blocks with each block in a specific location
6	move four blocks to a specified location (scalability study on L(A))
7	build a stack of 5 blocks in a defined order (scalability on I with I=5)
8	build a stack of 4 blocks in a defined order and unstack top block (scalability on I with I=4)
9	build a stack of 3 blocks in a defined order and unstack top 2 block (scalability on I with I=3)
10	build a stack of 2 blocks in a defined order, unstack and move top block of second stack (scalability on I with I=2)
11	build a stack of 4 blocks
12	build a stack of 3 blocks (scalability on Sigma with Sigma=3)
13	build a stack of 3 blocks, then unstack once (scalability on Sigma with Sigma=6)
14	build a stack of 3 blocks, then unstack twice (scalability on Sigma with Sigma=9)
15	build a stack of 3 blocks, then unstack 3 times (scalability on Sigma with Sigma=12)
16	build a stack of 3 blocks, then unstack 4 times (scalability on Sigma with Sigma=15)

7

Reward Machine Inference for Robotic Manipulation

"Wist je dat in bad zitten skrjim ne vorm van bespaering is?"
Flip Kowlier, "Kwestie van Organisatie"

Reward Machine Inference for Robotic Manipulation

M. Baert • S. Leroux • P. Simoens.
Presented at the AAAI workshop on generalization in planning (GenPlan), 2025.

Building on the work from the previous chapter, we relax the assumption of relying on an engineered feature representation that explicitly encodes all object positions. Instead, sub-goals are identified within a feature space automatically extracted by a deep neural network. We further demonstrate that reinforcement learning can be effectively used to train a policy for each sub-goal using the Reward Machine (RM) framework. Unlike previous RM inference strategies, our approach requires no predefined propositions or prior knowledge of the underlying sparse reward signals. Instead, it jointly learns the RM structure and identifies key high-level events that drive transitions between RM states. We validate our method on vision-based manipulation tasks, showing that the inferred RM accurately captures task structure and enables an RL agent to effectively learn an optimal policy.

7.1 Introduction

Combining Learning from Demonstrations (LfD) with Reinforcement Learning (RL) has empowered artificial agents to learn complex tasks from human examples in areas such as video games [1], board games [2], and robotics [3, 4]. LfD enhances RL by improving sample efficiency, which accelerates learning, and by reducing the effort required from human developers to manually specify precise task objectives. However, many LfD approaches struggle with long-duration tasks as they focus on directly learning a control policy from demonstrations [5, 6] or inferring a reward function [7, 8, 9]. Integrating abstract task structures, by decomposing complex tasks into manageable sub-tasks and providing structured guidance to the agent, has enabled RL to address long-horizon tasks more effectively [10, 11].

Reward machines (RMs) [12] provide a framework for defining such abstract task structures by encoding high-level task objectives in a structured, automata-inspired format. Originally RMs were developed to extend RL for environments with non-Markovian rewards, by enhancing the agent's state space with an abstract state layer that allow agents to retain memory of past actions. However, recent research [11] has demonstrated that RMs can also enhance performance in fully Markovian tasks by converting sparse task rewards into a denser reward signal, thereby offering more consistent guidance as the policy advances through abstract states. Despite their ad-

vantages, most current approaches require RMs to be manually specified by domain experts.

In this work, we address this gap by focusing on learning reward machines directly from visual demonstrations in robotic manipulation tasks. The RM framework depends on a set of propositions that encode high-level properties of the environment and a labeling function L that assigns truth values to these propositions based on the current environment state. Unlike prior approaches, which assume these propositions or feature detectors are predefined [13, 14, 15, 11], our approach jointly learns the RM structure and the mapping from environment states to Boolean propositions. Our method begins by capturing visual demonstrations. We leverage the observation that sub-goals are visited more frequently in expert demonstrations compared to other states [16, 17]. To identify these sub-goals, each frame is mapped to a low-dimensional feature vector. Through clustering, similar states are grouped and prototypical states (representing a certain sub-goal) can be identified. Finally, an RM is constructed capturing the sequential task structure from the demonstrations. This approach advances LfD by enabling the automatic inference of structured task representations, which can effectively guide RL agents in long-horizon tasks. We evaluate our method on a set of vision-based robotic manipulation tasks, demonstrating its effectiveness.

7.2 Related Work

There has been extensive research on integrating formal methods into RL. For example, several works adopt temporal logics to specify complex, long-horizon tasks [18, 19, 20, 21], while others investigate the use of automata, such as RMs, to formalize task specifications [22, 23, 12]. A common assumption in these approaches is that high-level knowledge, in the form of automata or temporal logic specifications, is available a priori. However, in real-world scenarios, this knowledge is often implicit and must be inferred from data.

Regarding the inference of temporal logics from data, many methods focus on learning temporal logic formulas using both positive and negative examples [24, 25]. However, in the context of LfD, only positive examples (i.e., demonstrations) are typically available. Some works address learning linear temporal logic (LTL) from only positive examples [26, 27], but these approaches usually assume that for each time step in the provided trajectories, the truth valuation of a set of Boolean propositions is known. More recent efforts aim to learn both a mapping from states to atomic propositions and the formula structure [17], although applying this in continuous state spaces

remains challenging.

In the domain of learning reward machines, most approaches focus on jointly learning both the RM and the policy through environment interaction. For example, discrete optimization can be used to learn an RM that decomposes the task into subproblems, such that combining their optimal memoryless policies yields an optimal solution for the original task [13]. Xu et al. [28] propose an iterative method that alternates between automaton inference, used to hypothesize RMs, and RL to optimize policies based on the current RM candidate. Inconsistencies between the hypothesis RM and observed trajectories then trigger re-learning. Verginis et al. [15] extend this approach to settings with partially known semantics. Additionally, Xu et al. [14] and Dohmen et al. [29] leverage active automata inference algorithms, such as the L* algorithm, to learn RMs. Although these approaches do not rely on expert demonstrations, they assume access to the reward function of the underlying MDP as well as the labeling function that maps states to propositions. Closest to our work is the method proposed by Camacho et al. [11], which learns RMs for vision-based robotic manipulation tasks. However, this approach also assumes access to a predefined mapping between low-level states and high-level propositions, limiting its applicability in more general settings where such a mapping is not readily available.

7.3 Background

7.3.1 Reinforcement Learning

A Markov decision process (MDP) [30] models sequential decision-making with the following components: a state space $\mathcal{S}$, an action space $\mathcal{A}$, a discount factor $\gamma \in [0, 1]$, a transition distribution $p(s' \mid s, a)$ describing the probability of reaching state s' from state s when taking action a, an initial state distribution $\mathcal{I}(s)$, and a reward function $R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, which defines a scalar reward for each state-action pair. An agent interacts with the environment at discrete timesteps t, generating trajectories $\tau = (s_0, ..., s_{T-1})$ of length T. The RL objective is to find an optimal policy π that maximizes the expected sum of discounted rewards: $\max_\pi \mathbb{E}_\pi \sum_{t=0}^{T-1} \gamma^t R(s_t, a_t)$.

In this work, we use off-policy deep Q-learning (DQN) [31] to train a policy π that selects actions by maximizing a Q-function: $\arg\max_{a \in \mathcal{A}} Q_\theta(s, a)$. Here, $Q_\theta(s, a)$, a neural network with parameters θ, estimates the expected cumulative reward for taking action a in state s. DQN updates θ by minimizing the temporal-difference (TD) error between the current Q-value estimate and a target value y_t. For each training step, transitions (s_t, a_t, r_t, s_{t+1}) are

sampled from a replay buffer. The loss function used is the Huber loss [32]:

$$\mathcal{L}(\theta) = \text{Huber}(Q_\theta(s_t, a_t) - y_t) \tag{7.1}$$

where

$$y_t = r(s_t, a_t, s_{t+1}) + \gamma \max_{a'} Q_\theta(s_{t+1}, a'). \tag{7.2}$$

Here, a' represents the set of all available actions.

7.3.2 Reward Machines

A Reward Machine (RM) [12] is defined as a tuple $\mathcal{R}_{AP,\mathcal{S},\mathcal{A}} = \langle U, u_0, \delta_u, \delta_r \rangle$, given a set of atomic propositions AP, a set of environment states $\mathcal{S}$, and a set of actions $\mathcal{A}$. Each proposition $p \in AP$ has a truth value of either true or false and represents a specific piece of information about the environment, such as object properties, agent statuses, or environmental conditions. The negation of a proposition p is denoted as $\neg p$. Propositions can be combined into more complex logical expressions using conjunctions ($\wedge$) and disjunctions ($\vee$). U is a finite set of states, with $u_0 \in U$ representing the initial state. The state-transition function δ_u maps pairs (u, AP) to new states in U, while the reward-transition function δ_r maps state transitions (u, u') to real-valued rewards in $\mathbb{R}$. At each time step t, the RM receives a truth assignment $\mathbf{p}_t$, which includes the propositions in AP that are true in the current environment state s_t. We can replace the standard reward in an MDP by an RM, creating a MDPRM. This requires a labeling function $L : \mathcal{S} \to 2^{|AP|}$ that assigns truth values to the propositions in AP based on the environment state. The state of the RM is updated every time step of the environment. If the RM is in state u and the agent takes action a to transition from environment state s to s', the RM transitions to state $u' = \delta_u(u, L(s'))$ and the agent receives a reward $r = \delta_r(u, u')$. A policy $\pi(s, u)$ for an MDPRM is conditioned both on the environment state and the RM state. This setup enables the modeling of non-Markovian reward functions within an MDPRM, as different histories of environment states can be distinguished by elements of a finite set of regular expressions over AP. Consequently, RMs can yield different rewards for identical environment transitions (s, a, s'), depending on the agent's prior state history.

7.4 Reward Machine Inference

In this section, we describe the process of inferring an RM from a set of high-dimensional demonstrations. The method consists of four steps:

[1] capturing demonstrations,

[2] extracting feature representations,

[3] inferring sub-goals through clustering,

[4] constructing the reward machine.

An overview of this process, applied to the task of building a predefined pyramid, is depicted in Figure 7.1.

7.4.1 Capturing Demonstrations

The first step is to capture a set of demonstrations from an expert performing the task. A camera with full observability of the workspace records these demonstrations, which results in a set of captured trajectories, denoted as $\mathcal{D} = \{\tau_0, \tau_1, \dots\}$, where each trajectory τ_i represents a sequence of observed states over time. Since an image frame contains all the necessary information for the agent to make decisions, we define a state $s_t \in \mathcal{S}$ as the image frame at time step t. Each state is represented as a tensor in $\mathbb{R}^{256\times256\times3}$, where the dimensions refer to the height, width, and RGB color channels of the image. Additionally, we capture the same demonstrations using a top-down camera, however, these are only utilized during the policy training phase (as described in the next section).

7.4.2 Extracting Feature Representations

To process the demonstrations, each frame s_t is cropped to retain only the most relevant portion of the workspace, such as the tabletop area where task-relevant objects are located (see Figure 7.1). After cropping, we extract a feature representation from each frame. This process transforms the image frames into a lower-dimensional feature space that retains the essential visual information. The extracted feature representation of a state s_t is denoted by $\phi(s_t)$.

7.4.3 Inferring Sub-Goals through Clustering

To identify sub-goals, we leverage the insight that true sub-goals occur more frequently in expert demonstrations than other states [16, 17]. Consequently, clustering can be used to detect high-density regions in the state space. Sub-goals are inferred by clustering the feature vectors obtained from the previous step, using the DBSCAN algorithm [33]. DBSCAN is robust to noise and does not require prior knowledge of the number of clusters,

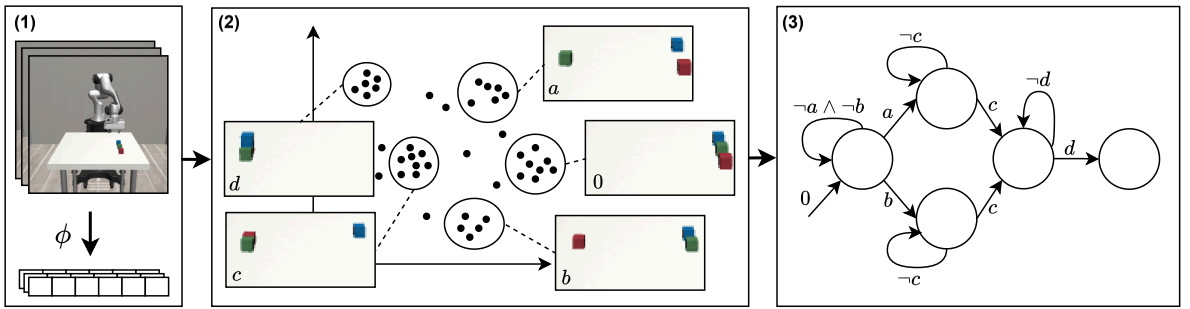


Figure 7.1: Overview of the proposed method applied to the task of building a predefined pyramid. (1) Visual demonstrations are captured, and feature embeddings are extracted using a pre-trained model ϕ. (2) Sub-goals are inferred by clustering the feature vectors obtained from the demonstrations. (3) An RM is constructed, capturing the valid temporal ordering and transitions between the inferred sub-goals. The 0-prototype corresponds with the initial RM state.

which is crucial as the number of sub-goals in the task is unknown. Let $\phi(\mathcal{D}) = \phi(s) \mid s \in \tau, \tau \in \mathcal{D}$ represent the set of all feature vectors extracted from all demonstrations. DBSCAN is applied to $\phi(\mathcal{D})$, yielding a set of k clusters $\{C_0, C_1, \ldots, C_{k-1}\}$. The cluster center for each cluster C_i is defined as μ_i:

$$\mu_i = \frac{1}{\mid C_i \mid} \sum_{\phi(s) \in C_i} \phi(s). \tag{7.3}$$

For each cluster C_i, we identify a prototypical state s_i^*, which is the state whose feature vector is closest to the cluster center:

$$s_i^* = \arg\min_{s \in C_i} \mid\mid \phi(s) - \mu_i \mid\mid_2 . \tag{7.4}$$

Here, $\mid\mid \cdot \mid\mid_2$ represents the Euclidean norm. Although clustering is performed in feature space, each prototypical state corresponds to an interpretable image, providing a human-understandable representation of each sub-goal.

7.4.4 Constructing the Reward Machine

The set of prototypical states forms the basis for defining the reward machine states. Let $U = \{u_0, u_1, \ldots, u_{k-1}\}$ represent the set of RM states, where each state u_i corresponds to a prototypical state s_i^*. We build the set of atomic propositions AP by defining a proposition p_i for each prototypical state s_i^*. The truth evaluation of each proposition $p_i \in AP$ is determined based on the Euclidean distance between the feature representation of the current state $\phi(s)$ and the feature representation of the corresponding prototypical state $\phi(s_i^*)$. Formally, for a given proposition $p_i \in AP$, corresponding to the sub-goal represented by prototypical state s_i^* and RM state u_i, the proposition is true if the Euclidean distance between $\phi(s)$ and $\phi(s_i^*)$ is less than a predefined threshold κ. The labeling function can then be expressed as:

$$L(s) = \{p_i \mid ||\phi(s) - \phi(s_i^*)||_2 < \kappa, \forall p_i \in AP\}. \tag{7.5}$$

An abstract demonstration $\hat{d}$ associated to a demonstration d can be defined as a sequence of sets of propositions $\mathbf{p}_0, \mathbf{p}_1, ..., \mathbf{p}_{n-1}$, where $\mathbf{p}_t$ is the abstraction of state s_t into AP by L, for each $t \in \{0, \ldots, T-1\}$. The set of abstract demonstrations $\hat{\mathcal{D}}$, thus, defines the directed connections between RM states. Transitions between abstract states observed in the demonstration should be reflected into the state-transition function δ_u. The inference of the state transition function δ_u is formalized in Algorithm 7.1. Given the definition of the labeling function (Eq. 7.5), multiple propositions may hold true for a single state. However, this would imply that the RM occupies multiple states simultaneously, which is not feasible. To avoid this, the

Algorithm 7.1: Inferring the state transition function δ_u from the set of abstract demonstrations $\hat{\mathcal{D}}$

Input: set of abstract demonstrations $\hat{\mathcal{D}}$

$\delta_u(u, \cdot) \leftarrow$ undefined $\quad \forall u \in U$

for $\hat{d} \in \hat{\mathcal{D}}$ **do**
  $u \leftarrow u_0$
  for $\mathbf{p} \in \hat{d}$ **do**
    for $p_i \in \mathbf{p}$ **do**
      if $\delta_u(u, p_i) = \textit{undefined}$ **then**

$$\delta_u(\hat{u}, \hat{p}) \leftarrow \begin{cases} u_i & \text{if } \hat{u} = u \text{ and } \hat{p} = p_i \\ \delta_u(\hat{u}, \hat{p}) & \text{otherwise} \end{cases}$$

      end
      $u \leftarrow \delta_u(u, p_i)$
    end
  end
end
return δ_u

hyperparameter κ should be tuned so that, for each state s_t, at most one proposition is true.

Next, we need to define the reward transition function δ_r to guide the agent toward reaching the goal states. A naive approach would be to assign a reward of 1 when the agent reaches a goal state in the RM and 0 otherwise. However, this creates a highly sparse reward signal, making it difficult for the agent to learn efficiently. To address this, we use potential-based reward shaping [34], which helps to construct a denser reward function while maintaining the same optimal behavior as the original sparse reward. The idea is to provide intermediate rewards that encourage progress toward the goal, making training more efficient. Inspired by [11], we define the reward function as follows:

$$\Psi(u) = \gamma^{d_{\text{goal}}(u)} \tag{7.6}$$

$$\delta_r(u, u') = \delta'_r(u, u') + \gamma\Psi(u') - \Psi(u), \tag{7.7}$$

where Ψ is the potential function, d_{goal} represents the shortest distance from the current RM state u to the goal state in terms of edges and δ'_r is the original sparse reward function. This shaped reward has the property that the reward is negative when the agent moves away from the goal in the RM graph, is slightly less negative when the agent stays in the same state, is

zero when the agent moves closer to the goal and evaluates to 1 when the agent reaches the goal state.

7.5 DQN on the Inferred Reward Machine

The action space is defined by a pick-and-place primitive, parameterized by four pixel coordinates: two for the pick location $(u_{\text{pick}}, v_{\text{pick}})$ and two for the place location $(u_{\text{place}}, v_{\text{place}})$. Through camera calibration and depth measurements, each pixel coordinate is mapped accurately to the robot's coordinate system, ensuring precise execution of these actions. For RM inference, images are captured from a front-view camera, while control actions use a top-down view of the workspace. The top-down perspective aligns directly with the workspace plane, reducing distortion and occlusions and facilitating accurate pixel-to-coordinate mappings. In contrast, front-view images are used for inferring sub-goals, as they avoid occlusions caused by the robot and provide a fuller view of the workspace.

Inspired by prior work in robotic manipulation [11, 35], we define our deep Q-function $Q_\theta(s, a)$ using two fully convolutional networks (FCNs) [36], ψ_{pick} and ψ_{place}. Each FCN receives a top-down image and outputs a dense, pixel-wise map of Q-values for all pick-and-place actions. To stabilize training, we only consider the Q-values inside a rectangular region corresponding to the table area, as picking or placing outside this region is irrelevant. Both FCNs share the same architecture based on ResNet50 [37] pre-trained on ImageNet [38]. We use an intermediate feature map from the output of the second residual block, then pass it through two convolutional layers that reduce the channels to one. Finally, the output is bilinearly upsampled to match the input resolution. During training, we compute gradients only for the Q-value of the executed action's pixel location, backpropagating zero loss for all other pixels. All parameters, including those in the pre-trained ResNet layers, are updated.

Following the DQRM approach [12], we train a discrete Q-function per RM state, each with a separate experience buffer for storing state-specific interactions and demonstrations. At the end of each episode, we update all Q-functions by sampling batches from their corresponding buffers. The input to the pick-action network ψ_{pick} is the current observation, while for ψ_{place}, we pass a top-down view prototype for each valid transition from the current RM state. This allows the network to focus on areas where blocks are expected in the next state, aiding in goal-oriented placement. Figure 7.2 depicts an overview of our DQN formulation applied to the pyramid task introduced earlier. Figure 7.2 is captured in the beginning of an episode with each block in its initial position and the RM in its initial state. There

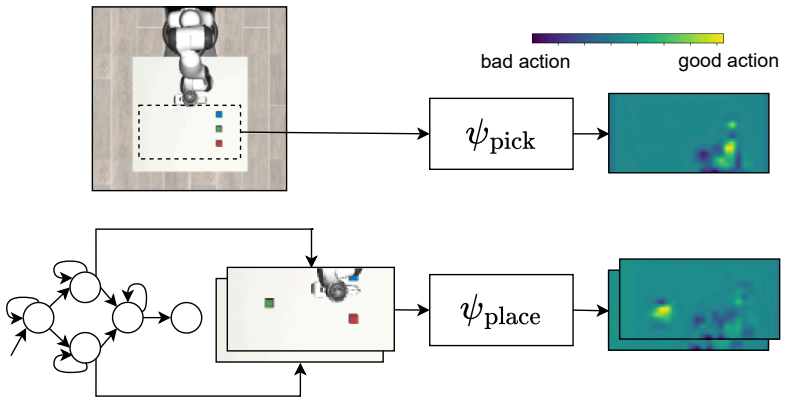


Figure 7.2: Overview of our DQN formulation. Each action consists of a picking operation at pixel coordinates $(u_{\text{pick}}, v_{\text{pick}})$ followed by a placing operation at $(u_{\text{place}}, v_{\text{place}})$. The Q-function $Q_\theta(s, a)$ is modeled using two FCN's: ψ_{pick} and ψ_{place} which generate pixel-wise Q-value maps for their respective actions.

are two valid transitions from the initial RM state, resulting in the place network ψ_{place} receiving two corresponding top-down prototypes. The Q-value maps in Figure 7.2 reveal that the pick network assigns high Q-values to the locations of the red and green blocks, indicating these are favorable pick locations. Similarly, the place network predicts high Q-values at the designated goal locations for each block.

7.6 Results

We evaluated our method on object manipulation tasks using unstructured human demonstrations within a simulated environment, employing a Franka Emika Panda robot in the robosuite simulation framework [39]. Five block-based manipulation scenarios were designed: Stack-2, where two blocks are stacked in a predefined order; Place-2, placing two blocks at specific locations; Pyramid-3, building a predefined three-block pyramid (Figure 7.1); Stack-3 (Figure 7.4) and Place-3 (Figure 7.3), similar to Stack-2 and Place-2, but involving three blocks. The number of demonstrations collected varied with task complexity. For example, Stack-3 required only a single demonstration due to the unique path from the initial to terminal RM state. In contrast, Place-3 required six demonstrations to cover all possible variations in sub-goal sequences. Demonstrations were generated by controlling the robot through an algorithmically defined expert policy, ensuring high-quality demonstrations that fully represent task variations. For RM

Table 7.1: Tuned parameters and the number of demonstrations used for the different scenarios.

	κ	$\epsilon_{\text{cluster}}$	*min points*	# demonstrations
Stack-2	3.0	0.8	15	1
Place-2	1.9	1.0	20	2
Pyramid-3	2.3	1.2	30	2
Stack-3	3.0	1.8	40	1
Place-3	1.9	1.7	110	6

inference, the feature extractor ϕ is parameterized by a ResNet-50 model [37] pre-trained on ImageNet [38]. DQN on the inferred RM was conducted with a fixed batch size of 16, using the Adam optimizer [40] and a learning rate of 0.0001. An ϵ-greedy exploration strategy is used, with ϵ initialized at 0.7 and decaying exponentially to 0.1 over training. DBSCAN requires two parameters: $\epsilon_{\text{cluster}}$, which defines the maximum distance between two points for one to be considered in the neighborhood of the other, and *min points*, the minimum number of points needed to establish a dense region. For each scenario, we tuned these parameters along with the threshold κ for matching states with prototypes. Table 7.1 provides an overview of the tuned parameters.

Our method successfully inferred the correct RMs across all tasks, producing meaningful prototype states representing the demonstrated task. The inferred RM accommodates variations in sub-goal sequences observed in the demonstrations by representing each possible ordering as an alternative route through the RM states. Figure 7.3 shows the inferred RM and prototypes for the Place-3 task. As presented in the previous sections, each RM state corresponds to a unique high-level proposition that triggers a transition. The (0)-prototype represents the initial RM state, with each subsequent prototype corresponding to a specific task sub-goal. Figure 7.1 and Figure 7.4 depict the inferred RM and prototypes for the Pyramid-3 and the Stack-3 scenarios respectively.

To quantitatively assess our approach, we report both total reward per episode and placement error, measured as the average distance between each object and its goal position (Figure 7.5). The total reward indicates if an agent effectively learns an optimal policy for the inferred RM. The placement error assesses the accuracy of the inferred RM, particularly in terms of its defined sub-goals.

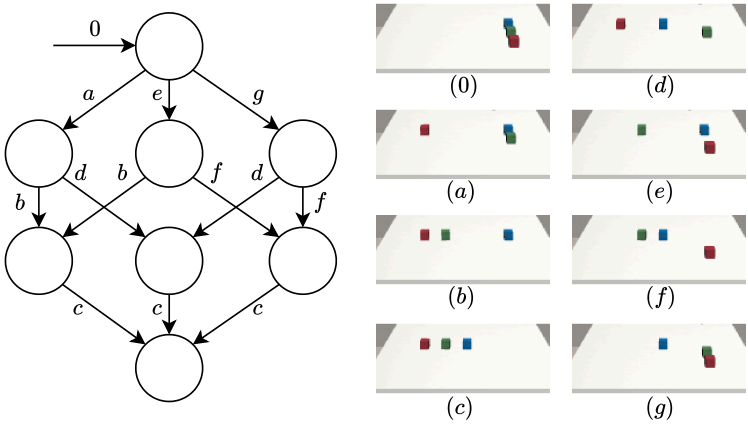


Figure 7.3: Inferred RM for the Place-3 task (left) and the corresponding state prototypes (right). We omitted the self-loop transitions for clarity.

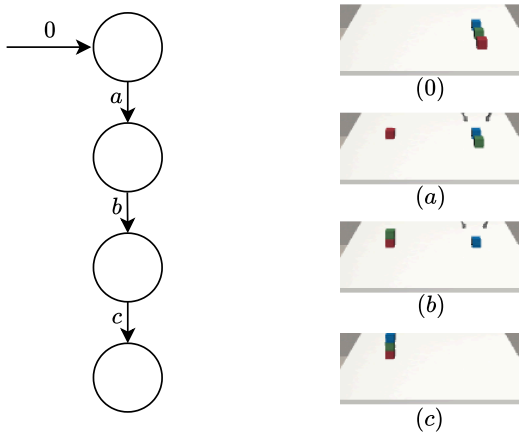
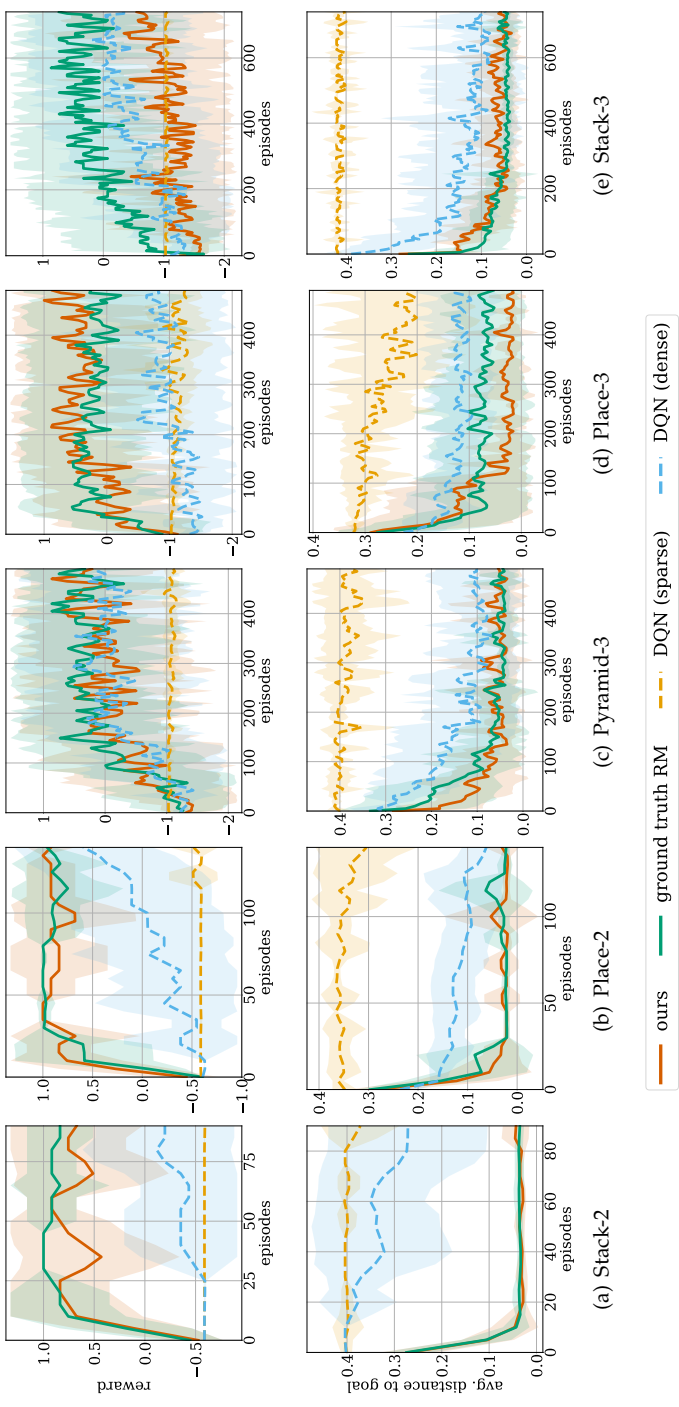


Figure 7.4: Inferred RM for the Stack-3 task (left) and the corresponding state prototypes (right). We omitted the self-loop transitions for clarity.

We compared our method to an agent using a ground truth RM with predefined propositions. For this ground truth RM, the set of propositions is again defined such that for each state, one proposition becomes true. Each proposition is defined by the ground truth object positions corresponding to that sub-goal. A proposition becomes true if the placement error between the proposition's object positions and the current object positions is below 0.05m. It is important to note that the ground truth reward machine can only be used if the location of each object can be determined at every time step. Fortunately, this condition is met in our simulation environment. To evaluate the impact of the RM, we compare its performance against an agent trained using vanilla DQN. Specifically, we differentiate between an agent that leverages the shaped reward provided by the RM (i.e., dense reward) and an agent that only receives a reward upon successful completion of the entire task (i.e., sparse reward). To track performance during training, we sample a trajectory from the current greedy policy ($\epsilon = 0$) every five episodes, measuring both the total accumulated reward and the placement error.

In achieving the maximum total reward of 1 per episode, agents must consistently transition towards and reach the final goal state in the RM graph (see Eq. 7.7). For tasks with two blocks higher total reward levels were reached compared to three-block tasks. Both DQRM agents exhibit similar learning curves across most environments. Placement error comparisons also show parity between our inferred RM and the ground truth RM agent, suggesting that any remaining placement error is primarily due to the control algorithm handling the pick-and-place actions. All methods show notable fluctuations in reward and placement error, attributed to noise in executing robot primitives. In our approach, additional variance is introduced by prototype matching, where variations in feature embeddings occasionally prevent RM transitions despite correct block placements. This explains the lower rewards in the Stack-3 task, despite low placement errors, certain RM transitions fail to trigger, preventing the final reward from being obtained even though the task is completed correctly. In the Place-3 task, the inferred RM agent outperformed the ground truth RM agent in both placement error and reward. This is due to the latter failing to recover the optimal policy in two runs. The DQN (dense) agent typically converges to a suboptimal policy, resulting in lower rewards and higher placement errors. Unlike DQRM, which learns a policy for each RM state, vanilla DQN must learn a single policy for the entire task, making the problem significantly more complex. This difficulty is reflected in both the obtained reward and distance error. Since the DQN (dense) agent adopts the same shaped reward as the DQRM agent, this experiment can be seen as an ablation study on



(a) Stack-2 (b) Place-2 (c) Pyramid-3 (d) Place-3 (e) Stack-3

Figure 7.5: Total reward obtained during one episode (top) and average distance between each object and its ground goal location, i.e., placement error (bottom) during training. Results are averaged over 10 runs, the shaded region represents the standard deviation.

the role of abstract state information in DQRM. Finally, in all cases, the DQN (sparse) agent fails to learn anything due to the large policy search space and the lack of guiding rewards. This supports the claim made by Camacho et al.[11] that RMs can significantly improve sample efficiency even in Markovian tasks.

7.7 Conclusion

We presented a novel method for inferring RMs from unstructured demonstrations, using video recordings as input. Unlike existing approaches, our method does not rely on predefined propositions. Instead, it uses clustering to derive meaningful sub-goals represented by prototypical states. By measuring the distance in feature space between the current observation and each prototype, our method detects sub-goal completion and enables state transitions within the RM. Experimental results show that our approach accurately infers the ground-truth RM with interpretable prototypes. Furthermore, the inferred RM enables an RL agent to learn an optimal policy, achieving similar placement accuracy to agents with access to ground-truth object positions and RMs.

Currently, our method converges on a single policy path between the initial and goal states, even when multiple valid paths exist in the RM. A promising direction for future work is to develop agents that can adaptively select alternative paths, which would improve robustness in environments where certain paths become infeasible due to perturbations (e.g., a block becomes unavailable). An interesting body of work in this direction is maximum entropy RL [41]. Additionally, enhancing prototype quality and detection accuracy could be achieved by exploring alternative embedding types, such as object-centric representations [42] that are likely more suited to robotic manipulation tasks. This could also facilitate the use of more complex types of objects. Integrating multi-camera views, or projecting views (e.g., from front to top), could further enrich feature representations, especially for real-world scenarios where camera placement may be limited. Finally, exploring the application of this technique in other domains, such as navigation, could extend its impact beyond manipulation tasks.

References

[1] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, et al. *Deep q-learning from demonstrations*. In Proceedings of the AAAI conference on artificial intelligence, volume 32, 2018.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. *Mastering the game of Go with deep neural networks and tree search*. nature, 529(7587):484–489, 2016.

[3] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. *A survey of robot learning from demonstration*. Robotics and autonomous systems, 57(5):469–483, 2009.

[4] P. Abbeel, A. Coates, and A. Y. Ng. *Autonomous helicopter aerobatics through apprenticeship learning*. The International Journal of Robotics Research, 29(13):1608–1639, 2010.

[5] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel. *Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation*. In 2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018, pages 1–8. IEEE, 2018. Available from: https://doi.org/10.1109/ICRA.2018.8461249, doi:10.1109/ICRA.2018.8461249.

[6] A. Mandlekar, D. Xu, R. Martín-Martín, S. Savarese, and L. Fei-Fei. *GTI: Learning to Generalize across Long-Horizon Tasks from Human Demonstrations*. In M. Toussaint, A. Bicchi, and T. Hermans, editors, Robotics: Science and Systems XVI, Virtual Event / Corvalis, Oregon, USA, July 12-16, 2020, 2020. Available from: https://doi.org/10.15607/RSS.2020.XVI.061, doi:10.15607/RSS.2020.XVI.061.

[7] J. Ho and S. Ermon. *Generative adversarial imitation learning*. Advances in neural information processing systems, 29, 2016.

[8] P. Abbeel and A. Y. Ng. *Apprenticeship learning via inverse reinforcement learning*. In Proceedings of the twenty-first international conference on Machine learning, page 1, 2004.

[9] M. Baert, P. Mazzaglia, S. Leroux, and P. Simoens. *Maximum Causal Entropy Inverse Constrained Reinforcement Learning*. arXiv preprint arXiv:2305.02857, 2023.

[10] M. Baert, S. Leroux, and P. Simoens. *Learning Task Specifications from Demonstrations as Probabilistic Automata*. arXiv preprint arXiv:2409.07091, 2024.

[11] A. Camacho, J. Varley, A. Zeng, D. Jain, A. Iscen, and D. Kalashnikov. *Reward machines for vision-based robotic manipulation*. In 2021 IEEE International Conference on Robotics and Automation (ICRA), pages 14284–14290. IEEE, 2021.

[12] R. T. Icarte, T. Q. Klassen, R. Valenzano, and S. A. McIlraith. *Reward machines: Exploiting reward function structure in reinforcement learning*. Journal of Artificial Intelligence Research, 73:173–208, 2022.

[13] R. Toro Icarte, E. Waldie, T. Klassen, R. Valenzano, M. Castro, and S. McIlraith. *Learning reward machines for partially observable reinforcement learning*. Advances in neural information processing systems, 32, 2019.

[14] Z. Xu, B. Wu, A. Ojha, D. Neider, and U. Topcu. *Active finite reward automaton inference and reinforcement learning using queries and counterexamples*. In Machine Learning and Knowledge Extraction: 5th IFIP TC 5, TC 12, WG 8.4, WG 8.9, WG 12.9 International Cross-Domain Conference, CD-MAKE 2021, Virtual Event, August 17–20, 2021, Proceedings 5, pages 115–135. Springer, 2021.

[15] C. K. Verginis, C. Koprulu, S. Chinchali, and U. Topcu. *Joint learning of reward machines and policies in environments with partially known semantics*. Artificial Intelligence, page 104146, 2024.

[16] B. Ghazanfari, F. Afghah, and M. E. Taylor. *Sequential association rule mining for autonomously extracting hierarchical task structures in reinforcement learning*. IEEE Access, 8:11782–11799, 2020.

[17] M. Baert, S. Leroux, and P. Simoens. *Learning Temporal Task Specifications From Demonstrations*. In International Workshop on Explainable, Transparent Autonomous Agents and Multi-Agent Systems, pages 81–98. Springer, 2024.

[18] X. Li, C.-I. Vasile, and C. Belta. *Reinforcement learning with temporal logic rewards*. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 3834–3839. IEEE, 2017.

[19] Y.-L. Kuo, B. Katz, and A. Barbu. *Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of LTL formulas*. In 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 5604–5610. IEEE, 2020.

[20] Z. Xiong, J. Eappen, A. H. Qureshi, and S. Jagannathan. *Constrained Hierarchical Deep Reinforcement Learning with Differentiable Formal Specifications*. 2022.

[21] C. Voloshin, H. Le, S. Chaudhuri, and Y. Yue. *Policy optimization with linear temporal logic constraints*. Advances in Neural Information Processing Systems, 35:17690–17702, 2022.

[22] B. Araki, K. Vodrahalli, T. Leech, C.-I. Vasile, M. D. Donahue, and D. L. Rus. *Learning to plan with logical automata*. 2019.

[23] B. Araki, X. Li, K. Vodrahalli, J. DeCastro, M. Fry, and D. Rus. *The logical options framework*. In International Conference on Machine Learning, pages 307–317. PMLR, 2021.

[24] Z. Kong, A. Jones, and C. Belta. *Temporal logics for learning and detection of anomalous behavior*. IEEE Transactions on Automatic Control, 62(3):1210–1222, 2016.

[25] G. Bombara and C. Belta. *Offline and online learning of signal temporal logic formulae using decision trees*. ACM Transactions on Cyber-Physical Systems, 5(3):1–23, 2021.

[26] A. Shah, P. Kamath, J. A. Shah, and S. Li. *Bayesian inference of temporal task specifications from demonstrations*. Advances in Neural Information Processing Systems, 31, 2018.

[27] R. Roy, J.-R. Gaglione, N. Baharisangari, D. Neider, Z. Xu, and U. Topcu. *Learning interpretable temporal properties from positive examples only*. In Proceedings of the AAAI Conference on Artificial Intelligence, volume 37, pages 6507–6515, 2023.

[28] Z. Xu, I. Gavran, Y. Ahmad, R. Majumdar, D. Neider, U. Topcu, and B. Wu. *Joint inference of reward machines and policies for reinforcement learning*. In Proceedings of the International Conference on Automated Planning and Scheduling, volume 30, pages 590–598, 2020.

[29] T. Dohmen, N. Topper, G. Atia, A. Beckus, A. Trivedi, and A. Velasquez. *Inferring probabilistic reward machines from non-markovian reward signals for reinforcement learning*. In Proceedings of the International Conference on Automated Planning and Scheduling, volume 32, pages 574–582, 2022.

[30] R. Bellman. *A Markovian decision process*. Journal of mathematics and mechanics, pages 679–684, 1957.

[31] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. *Human-level control through deep reinforcement learning*. nature, 518(7540):529–533, 2015.

[32] P. J. Huber. *Robust estimation of a location parameter*. In Breakthroughs in statistics: Methodology and distribution, pages 492–518. Springer, 1992.

[33] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. *A density-based algorithm for discovering clusters in large spatial databases with noise*. In kdd, volume 96, pages 226–231, 1996.

[34] A. Y. Ng, D. Harada, and S. Russell. *Policy invariance under reward transformations: Theory and application to reward shaping*. In Icml, volume 99, pages 278–287, 1999.

[35] A. Zeng, S. Song, K.-T. Yu, E. Donlon, F. R. Hogan, M. Bauza, D. Ma, O. Taylor, M. Liu, E. Romo, et al. *Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching*. The International Journal of Robotics Research, 41(7):690–705, 2022.

[36] J. Long, E. Shelhamer, and T. Darrell. *Fully convolutional networks for semantic segmentation*. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3431–3440, 2015.

[37] K. He, X. Zhang, S. Ren, and J. Sun. *Deep residual learning for image recognition. CoRR abs/1512.03385 (2015)*, 2015.

[38] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. *Imagenet: A large-scale hierarchical image database*. In 2009 IEEE conference on computer vision and pattern recognition, pages 248–255. Ieee, 2009.

[39] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, and Y. Zhu. *robosuite: A Modular Simulation Framework and Benchmark for Robot Learning*. In arXiv preprint arXiv:2009.12293, 2020.

[40] D. P. Kingma. *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980, 2014.

[41] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*. In International conference on machine learning, pages 1861–1870. PMLR, 2018.

[42] F. Locatello, D. Weissenborn, T. Unterthiner, A. Mahendran, G. Heigold, J. Uszkoreit, A. Dosovitskiy, and T. Kipf. *Object-centric learning with*

slot attention. Advances in neural information processing systems, 33:11525–11538, 2020.

8

Conclusions and Future Research Directions

"Blue are the life-giving waters, taken for granted, they quietly understand"
Jimi Hendrix, "Bold as Love"

The primary goal of this Ph.D. dissertation was to develop methods for learning symbolic representations from demonstrations that capture the behavioral patterns of agents. This overarching problem was approached from multiple perspectives. The dissertation was organized into two main parts: the first focuses on learning constraints that describe restrictions on agent behavior (e.g., rules, norms, and safety constraints), while the second explores learning high-level orderings of sub-goals. Table 8.1 summarizes the key distinctions between the methods proposed across the different chapters of this thesis. The comparison considers the form of the learned representation, the type of specifications captured, support for continuous state spaces, the structure of the input observations (e.g., interpretable features), and the inductive biases underlying each approach. This chapter is organized as follows: Sections 8.1 and 8.2 present conclusions and future directions for Parts 1 and 2, respectively. Section 8.3 outlines broader future research directions beyond the scope of the two main parts.

Table 8.1: Meta overview of the methods introduced in the different chapters of this thesis.

Chapter	Representation	Learned Specification	State Space	Structured Observations	Inductive Bias
ch. 2	Neural Network	Constraints	Continuous	no	MDP, Continuity
ch. 3	First-Order Logic	Constraints	Discrete	yes	MDP, Known Transition Dynamics, Background Knowledge, Search Bias, Labeling Function
ch. 4	Propositional Logic	Constraints	Continuous	yes	No Negative Evidence Bias, Semantic Features
ch. 5	LTL	Constraints, Sequential Task Structure	Discrete	no	MDP, Search Bias
ch. 6	PDFA	Sequential Task Structure, User Preferences	Continuous	yes	Semantic Features, Search Bias, Low-Level Skills
ch. 7	RM	Sequential Task Structure	Continuous	no	MDP

8.1 Inverse Constrained Reinforcement Learning

8.1.1 Conclusions

Traditionally, ICRL assumes that the environment is modeled as a CMDP. Given the reward function, the MDP structure can be leveraged to identify the constraints. This principle underpins the methods discussed in Chapters 2 and 3, where the use of a maximum entropy objective acts as a regularization mechanism to avoid overfitting to expert demonstrations. In Chapter 2, we introduced a novel ICRL objective grounded in the principle of maximum causal entropy. Through theoretical analysis and empirical evaluation, we demonstrated the ability of this method to learn constraints in complex environments characterized by continuous state spaces and stochastic transition dynamics. However, the cost function, represented by a neural network, suffers from a lack of interpretability. While the learned policies effectively adhered to the ground truth constraints, further analysis revealed that the recovered cost function did not precisely capture the underlying ground truth constraints, highlighting a gap between constraint satisfaction and accurate constraint representation. We attribute this discrepancy to the reliance on feature matching between nominal and expert policies, as opposed to directly inferring the most likely constraints, as in Chapter 3. In Chapter 3, we introduced a method that infers constraints as first-order logic formulas using predicates grounded in the MDP's state space. Although this approach provides high interpretability, its applicability is limited to simple environments. This is due to the assumption of a discrete state space, known transition dynamics, a human engineered search bias and background knowledge and the availability of a labeling function mapping logic predicates to the environment's state-action space. Chapter 4 explores an alternative approach using one-class decision trees, framing constraint inference as a one-class classification problem. This method provides interpretability by learning a decision tree that describes the distribution of the demonstrations, under the assumption that the semantics of the state-feature vector are known. However, it tends to overfit to the expert demonstrations as it does not leverage the reward function to distinguish between unvisited states due to constraints versus low task value. For this method, behavior which is not similar to the expert demonstrations is assumed to be invalid. In summary, the method in Chapter 3 excels in inferring highly interpretable logic rules but is restricted to simple environments. The approach in Chapter 2 scales to complex environments but sacrifices interpretability. Meanwhile, the method in Chapter 4 allows for inferring interpretable constraints in more complex environments but is prone to overfitting due to its lack of integration with reward functions.

8.1.2 Future Work

As discussed, each method presented in this dissertation has its own limitations. To address these shortcomings, one potential direction is to combine elements from individual methods to leverage their respective strengths. For instance, building on the approach introduced in Chapter 2, the neural cost function could be replaced with an inherently interpretable classifier, such as a decision tree. An important consideration in this approach is to retain a copy of the learned tree at each iteration. The collection of trees generated throughout training would form a forest, collectively representing the constraints. This ensures that constraints learned in earlier iterations are not lost, as it is not guaranteed that a tree learned at iteration 10 will still preserve the constraints identified at iteration 1. We also hypothesize that the method introduced in Chapter 3 could be extended to more realistic environments. Instead of explicitly computing state visitation frequencies, these could be approximated using Monte Carlo sampling. Additionally, deep reinforcement learning techniques could be adopted to update the policy at each iteration.

For constraint learning, experiments were conducted in gridworlds, simulated robotics environments, and autonomous driving scenarios. Most of these environments function as benchmark tasks to allow for straightforward comparisons between methods. It would be interesting to explore how these methods perform in more realistic and dynamic environments with greater complexity. For instance, testing constraint learning in real-world robotic platforms or traffic simulations with human-in-the-loop interactions could provide valuable insights into scalability and practical applicability. Additionally, integrating constraints learned in simulation into real-world systems would be a critical step forward.

Another avenue for future exploration addresses the reliance on these simulated environments for training agents. Simulated environments provide a safe and controlled space where agents can iteratively refine their behaviors through exploration and feedback. These settings allow agents to experiment freely, even violating constraints to better understand their boundaries and limitations. However, when a simulation environment is unavailable, the risks associated with constraint violations during training become a significant concern. Violations in real-world settings can lead to harmful consequences, unsafe conditions, or costly failures. Future work could focus on advancing ICRL methods to ensure safety during the learning process, even in the absence of a simulation environment. Developing approaches that prioritize safety while maintaining effective learning could open new possibilities for deploying ICRL in real-world, high-stakes scenarios.

8.2 Learning Plans from Demonstration

8.2.1 Conclusions

In the second part of my dissertation, I focused on developing algorithms to learn representations of high-level plans from demonstrations. In Chapter 5, we introduced a method for inferring temporal logic formulas that capture high-level task structures. This method builds on the same intuition as the approach in Chapter 3, iteratively extending a specification to maximize the posterior probability of the demonstrations. This method benefits from the expressiveness of LTL, which allows for representing complex tasks that include both constraints and sequential structure. However, its reliance on state enumeration limits practicality in continuous state spaces. In Chapter 6, we narrow our focus to robotic object manipulation tasks, assuming each task can be decomposed into a sequence of sub-goals. We developed an algorithm to learn both the temporal ordering of these sub-goals and the demonstrator's preferences, represented as a PDFA. This approach leverages the insight that sub-goals are shared across all demonstrations. Hence, they can be identified by detecting high-density regions in the state feature space using clustering techniques. In Chapter 7, we replaced engineered features with feature vectors extracted from a deep neural network to enhance scalability and generalization. Using the reward machine framework, we showed that it is possible to simultaneously infer task structure and learn an RL policy to accomplish the task.

8.2.2 Future Work

An exciting direction for future research is to extend the method presented in Chapter 7 to object manipulation tasks involving more complex and diverse objects. This could involve enhancing the task structure inference process to capture richer interactions, such as multi-object dependencies. Additionally, leveraging state-of-the-art feature extraction techniques could improve the representation of complex objects and their interactions. On the policy learning side, exploring advanced neural network architectures tailored for object manipulation [1] could further enhance the agent's ability to learn and execute sophisticated manipulation strategies. While RMs and PDFAs are capable of modeling constraints, our focus was limited to sequential task structure due to the demands of the targeted robotics application. Future work could extend these methods to fully exploit the expressiveness of RMs and PDFAs, including their ability to represent constraints. There are also considerable efforts aimed at scaling RMs to more complex, real-world environments. Hierarchical RMs [2], for instance, provide a natural framework to specify reward structures across different levels of abstraction,

facilitating more efficient learning in tasks with compositional structure. Additionally, Ardon et al. [3] demonstrate how RMs can be extended to relational domains by leveraging first-order logic. Other research extends the applicability of RMs to uncertain environments by introducing methods that accommodate noisy or unreliable labeling functions [4, 5].

8.3 General Future Perspectives

In this dissertation, we focused on "hard" rules, i.e., strict, deterministic definitions of behavior. However, uncertainty plays a pivotal role in both acquiring abstract knowledge about the world and defining flexible rules that govern behavior. This is especially true in real-world scenarios, where strict rules are uncommon, and exceptions often depend on the context. For instance, the rule "cars should not run red lights" is generally applicable but can be overridden for ambulances responding to emergencies. Such cases emphasize the need for designing systems capable of incorporating, representing, and reasoning with uncertainty in their models and rule definitions. Uncertainty naturally arises when systems attempt to learn abstract knowledge from noisy, incomplete, or ambiguous data. For example, an autonomous agent observing human behavior may struggle to infer definitive rules, as observed actions are often influenced by unobservable variables such as intent or contextual priorities. Accurately representing this uncertainty allows systems to generalize more effectively, reducing overconfidence in overly rigid or incorrect interpretations of observed behavior. Rather than enforcing binary decisions, rules should include probabilistic or conditional components that accommodate context. Future work should, therefore, explore approaches that embed uncertainty into rule representations. For instance, probabilistic rules can encode the likelihood of exceptions or contextual adjustments, providing a structured way to handle uncertainty. However, several open questions remain. While probabilistic rules offer flexibility, they may lack the clarity and interpretability of traditional symbolic rules. How can we balance interpretability with the need for uncertainty? Another challenge is learning exceptions from limited data. Rare events, such as ambulances running red lights, are inherently difficult to capture. How can models generalize effectively from sparse observations to infer robust and contextually appropriate exceptions?

Another particularly pressing challenge in neuro-symbolic methods is symbol grounding [6]. Symbol grounding refers to the process of translating raw, low-level information about the world into discrete, meaningful symbols that can be used in logical reasoning. This is a critical bottleneck that limits the broad applicability of logic-based approaches in real-world scenarios. In propositional logic, symbol grounding involves defining a labeling function

that, given an observation, evaluates the truth values of a predefined set of propositions. For example, in a robotic setting, the labeling function might determine whether a robot is "holding an object" or "in a restricted area". However, constructing such a function often requires significant domain knowledge, handcrafted features, or access to structured data, which is not always feasible in dynamic or complex environments. The methods introduced in Chapters 6 and 7 leverage clustering to establish a mapping between the low-level state space and a symbolic representation. Similarly, Aspis et al. [7] leverage clustering for neuro-symbolic reasoning tasks. With first-order logic, the symbol grounding problem becomes even more intricate. First-order logic extends propositional logic by enabling the representation of relationships between entities. Here, the grounding process must not only evaluate individual propositions but identify and map entities (e.g., objects, agents) and their relationships (e.g., "is-above", "is-near") from raw observations. There are already some works [8, 9] which use salient regions in demonstrations for learning first-order logic predicates. I am eager to see how future research will advance these clustering-based techniques for identifying symbols in unstructured data.

Recent developments in multimodal foundation models offer compelling opportunities to extend this line of work. In RL, foundation models are used to translate high-level textual goals into reward functions [10, 11] or sequences of sub-goals [12, 13]. Vision Language Action (VLA) models extend VLMs by coupling perception and language with embodied actions [14]. Despite these advances, large multimodal models remain to have challenges with interpretability, consistency, reasoning and planning [15, 16, 17]. I believe the future lies in hybrid architectures that combine the strengths of symbolic reasoning with the flexibility and perceptual grounding of foundation models. VLMs can enable a form of semantic grounding that supports the automatic extraction of symbolic predicates from unstructured visual data [18, 19, 20, 21, 22]. In these methods, a foundation model acts as a neural interface converting unstructured sensory input into symbolic forms usable by classical planners or logic-based systems. In this way, the labeling function which maps unstructured observations to symbolic predicates can be approximated by a foundation model, significantly reducing the need for manual feature engineering. However, like all foundation models, these models are prone to generating harmful or counterfactual responses. A promising avenue for addressing these limitations lies in the development of interactive algorithms that actively query humans to iteratively refine their grounding process. LMMs also show promise in translating natural language instructions into LTL formulas, enabling the specification of complex tasks in a human-interpretable and formally analyzable way [23, 22].

Beyond this, I see significant potential in leveraging foundation models to guide the search processes of symbolic methods. By tapping into the commonsense knowledge embedded in these models, we can improve the efficiency and relevance of symbolic reasoning.

References

[1] A. Mousavian, C. Eppner, and D. Fox. *6-dof graspnet: Variational grasp generation for object manipulation*. In Proceedings of the IEEE/CVF international conference on computer vision, pages 2901–2910, 2019.

[2] D. Furelos-Blanco, M. Law, A. Jonsson, K. Broda, and A. Russo. *Hierarchies of reward machines*. In International Conference on Machine Learning, pages 10494–10541. PMLR, 2023.

[3] L. Ardon, D. Furelos-Blanco, R. Parać, and A. Russo. *FORM: Learning Expressive and Transferable First-Order Logic Reward Machines*. arXiv preprint arXiv:2501.00364, 2024.

[4] A. Li, Z. Chen, T. Klassen, P. Vaezipoor, R. Toro Icarte, and S. McIlraith. *Reward Machines for Deep RL in Noisy and Uncertain Environments*. Advances in Neural Information Processing Systems, 37:110341–110368, 2024.

[5] R. Parac, L. Nodari, L. Ardon, D. Furelos-Blanco, F. Cerutti, and A. Russo. *Learning robust reward machines from noisy labels*. arXiv preprint arXiv:2408.14871, 2024.

[6] S. Harnad. *The symbol grounding problem*. Physica D: Nonlinear Phenomena, 42(1-3):335–346, 1990.

[7] Y. Aspis, K. Broda, J. Lobo, and A. Russo. *Embed2sym-scalable neuro-symbolic reasoning via clustered embeddings*. In Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning, volume 19, pages 421–431, 2022.

[8] N. Shah, J. Nagpal, P. Verma, and S. Srivastava. *From Reals to Logic and Back: Inventing Symbolic Vocabularies, Actions, and Models for Planning from Raw Data*. arXiv preprint arXiv:2402.11871, 2024.

[9] L. Keller, D. Tanneberg, and J. Peters. *Neuro-Symbolic Imitation Learning: Discovering Symbolic Abstractions for Skill Learning*. arXiv preprint arXiv:2503.21406, 2025.

[10] J. Rocamonde, V. Montesinos, E. Nava, E. Perez, and D. Lindner. *Vision-language models are zero-shot reward models for reinforcement learning*. arXiv preprint arXiv:2310.12921, 2023.

[11] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar. *Eureka: Human-level reward design via coding large language models*. arXiv preprint arXiv:2310.12931, 2023.

[12] Y. Shukla, W. Gao, V. Sarathy, A. Velasquez, R. Wright, and J. Sinapov. *LgTS: Dynamic Task Sampling using LLM-generated sub-goals for Reinforcement Learning Agents*. CoRR, abs/2310.09454, 2023. Available from: https://doi.org/10.48550/arXiv.2310.09454, arXiv:2310.09454, doi:10.48550/ARXIV.2310.09454.

[13] M. Dalal, T. Chiruvolu, D. Chaplot, and R. Salakhutdinov. *Plan-Seq-Learn: Language Model Guided RL for Solving Long Horizon Robotics Tasks*. 2024.

[14] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn, et al. *Rt-2: Vision-language-action models transfer web knowledge to robotic control*. arXiv preprint arXiv:2307.15818, 2023.

[15] K. Valmeekam, M. Marquez, S. Sreedharan, and S. Kambhampati. *On the planning abilities of large language models-a critical investigation*. Advances in Neural Information Processing Systems, 36:75993–76005, 2023.

[16] K. Stechly, K. Valmeekam, and S. Kambhampati. *Chain of thought-lessness? an analysis of cot in planning*. In The Thirty-eighth Annual Conference on Neural Information Processing Systems, 2024.

[17] K. Valmeekam, K. Stechly, A. Gundawar, and S. Kambhampati. *A Systematic Evaluation of the Planning and Scheduling Abilities of the Reasoning Model o1*. Transactions on Machine Learning Research, 2025.

[18] Z. Peng, W. Wang, L. Dong, Y. Hao, S. Huang, S. Ma, and F. Wei. *Kosmos-2: Grounding multimodal large language models to the world*. arXiv preprint arXiv:2306.14824, 2023.

[19] H. You, H. Zhang, Z. Gan, X. Du, B. Zhang, Z. Wang, L. Cao, S.-F. Chang, and Y. Yang. *Ferret: Refer and ground anything anywhere at any granularity*. arXiv preprint arXiv:2310.07704, 2023.

[20] Y. Liang, N. Kumar, H. Tang, A. Weller, J. B. Tenenbaum, T. Silver, J. F. Henriques, and K. Ellis. *Visualpredicator: Learning abstract world models with neuro-symbolic predicates for robot planning*. arXiv preprint arXiv:2410.23156, 2024.

[21] A. Athalye, N. Kumar, T. Silver, Y. Liang, T. Lozano-Pérez, and L. P. Kaelbling. *Predicate Invention from Pixels via Pretrained Vision-Language Models*. arXiv preprint arXiv:2501.00296, 2024.

[22] B. Quartey, E. Rosen, S. Tellex, and G. Konidaris. *Verifiably following complex robot instructions with foundation models*. arXiv preprint arXiv:2402.11498, 2024.

[23] J. X. Liu, Z. Yang, I. Idrees, S. Liang, B. Schornstein, S. Tellex, and A. Shah. *Grounding complex natural language commands for temporal tasks in unseen environments*. In Conference on Robot Learning, pages 1084–1110. PMLR, 2023.