Inverse Reinforcement Learning Through Logic Constraint Inference

Mattijs Baert[1*], Sam Leroux[1] and Pieter Simoens[1]

[1]IDLab, Department of Information Technology, Ghent University- imec, Technologiepark 126, Ghent, B-9052, Belgium.

*Corresponding author(s). E-mail(s): mattijs.baert@ugent.be;
Contributing authors: sam.leroux@ugent.be;
pieter.simoens@ugent.be;

Abstract

Autonomous robots start to be integrated in human environments where explicit and implicit social norms guide the behavior of all agents. To assure safety and predictability, these artificial agents should act in accordance with the applicable social norms. However, it is not straightforward to define these rules and incorporate them in an agent's policy. Particularly because social norms are often implicit and environment specific. In this paper, we propose a novel iterative approach to extract a set of rules from observed human trajectories. This hybrid method combines the strengths of inverse reinforcement learning and inductive logic programming. We experimentally show how our method successfully induces a compact logic program which represents the behavioral constraints applicable in a Tower of Hanoi and a traffic simulator environment. The induced program is adopted as prior knowledge by a model-free reinforcement learning agent to speed up training and prevent any social norm violation during exploration and deployment. Moreover, expressing norms as a logic program provides improved interpretability, which is an important pillar in the design of safe artificial agents, as well as transferability to similar environments.

Keywords: Inductive Logic Programming, Inverse Reinforcement Learning, Answer Set Programming, Constraint Inference, Constrained Markov Decision Process.

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. 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 [3–5]. 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 with 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. 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, ILP requires both positive and negative examples while we consider observations only consist of positive examples. 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 paper, 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 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 elaborates on our proposed method. The proposed method and the induced hypotheses are tested and evaluated in section 4. Section 5 provides an overview of related work. Concluding remarks and future directives are given in section 6.

2 Background

2.1 Constrained Markov Decision Process

A finite-state Markov decision process (MDP) is a model for sequential decision making defined as $\mathcal{M} = (S, \{A_s\}, \mathcal{G}, P_{\text{trans}}, P_0, R, \phi)$ [10]. S is a finite set of discrete states and A a finite set of discrete actions. $\{A_s\}$ denotes the set of sets of available actions for all states, 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 $g \in \mathcal{G}$:

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

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}.

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 sub-optimal 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.

2.3 Answer Set Programming

Answer set programming (ASP) is a declarative problem solving approach [13]. 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 head$(r) \in X$ whenever body$(r)^+ \subseteq X$ and 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. The reduct P^X of a program P relative to an interpretation X is defined as: $P^X = \{\text{head}(r)$:- body$(r)^+ \mid r \in P,$ body$(r)^- \cap X = \emptyset\}$. In this work, we adopt clingo [14] to ground and solve ASP programs.

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. $\forall\, e^+ \in E^+ : \exists\, A \in \text{AS}(B \cup H)$ s.t. A extends e^+.
3. $\forall\, e^- \in E^- : \nexists\, A \in \text{AS}(B \cup H)$ 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 Method

In this section we formally describe the proposed method. Figure 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

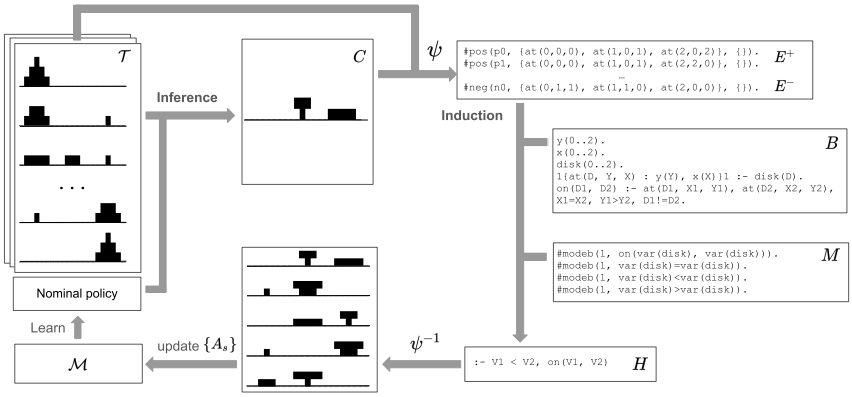


Fig. 1: Overview of a single iteration of the proposed method applied to the towers of Hanoi environment. First, an optimal policy is learned from the nominal MDP $\mathcal{M}$. During the inference step, a constraint is inferred from a set of trajectories $\mathcal{T}$ and the nominal policy. The constraint 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 $\{A_s\}$.

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 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. Subsection 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 3.2 presents the program

induction stage and the integration with constraint inference. The complete algorithm is depicted in algorithm 1.

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:

$$c^* = \arg\max_{c \in \mathcal{C}} P_{\mathcal{M}^{C \cup c}}(\mathcal{T}). \tag{2}$$

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 $\tau \in \mathcal{T}$:

$$P_{\mathcal{M}^C}(\mathcal{T}) = \frac{1}{Z(C)^{|\mathcal{T}|}} \prod_{\tau \in \mathcal{T}} e^{R(\tau)} \mathbb{1}^{\mathcal{M}^C}(\tau). \tag{3}$$

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 $\mathcal{M}^C$, $\xi_{\mathcal{M}^C}$:

$$Z(C) = \sum_{\tau \in \xi_{\mathcal{M}^C}} e^{R(\tau)}. \tag{4}$$

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 (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:

$$\begin{aligned} c^* &= \arg\max_{c \in \mathcal{C}} P_{\mathcal{M}}(\xi^-_{\mathcal{M}^{C \cup c}}) \\ &\text{s.t. } \mathcal{T} \cap \xi^-_{\mathcal{M}^{C \cup c}} = \emptyset. \end{aligned} \tag{5}$$

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:

$$D_{s',a'} = \sum_{g \in \mathcal{G}} D_{s',g} \pi(a' \mid s', g) \mathbb{1}^{\mathcal{M}^C}(s', a'). \tag{6}$$

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

$$D_{s',g} = \sum_{s \in S} \sum_{a \in A_s} D_{s,g} \pi(a \mid s, g) P_{trans}(s' \mid s, a). \tag{7}$$

The final objective then becomes:

$$c^* = \arg\max_{s,a} D_{s,a} \;\text{ s.t. }\; (s,a) \notin \mathcal{T}. \tag{8}$$

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]:

$$\pi(a \mid s, g) = \frac{Z_{s,a,g}}{Z_{s,g}} \tag{9}$$

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

$$Z_{s,a,g} = \sum_{s' \in S} Z_{s',g} P_{trans}(s' \mid s, a) e^{R(s,a,s',g)} \tag{10}$$

$$Z_{s,g} = \sum_{a \in A_s} Z_{s,a,g}. \tag{11}$$

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 [7, 8] 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.

$$R(s,a,s',g) = \begin{cases} 1 & \text{if } s' \in \mathcal{T}_T^g \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

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.

3.2 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. B and S_M should be provided. 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:

$$\psi(s,a) \in E^+ : (s,a) \in \mathcal{T} \tag{13}$$

$$\psi(s,a) \in E^- : (s,a) \in C. \tag{14}$$

An example of ψ for the towers of Hanoi environment can be found in Figure 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 1.

4 Experiments

In this section we perform an experimental evaluation of the proposed method. Subsection 4.1 provides a brief overview of the used environments. In subsection 4.2, a qualitative evaluation follows of the induced hypotheses. Subsection 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 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 empirical violation probability both during and after training. At last, in subsection 4.5 we inspect the transferability of the induced rules to other but similar environments.

Algorithm 1 Logic constraint inference algorithm

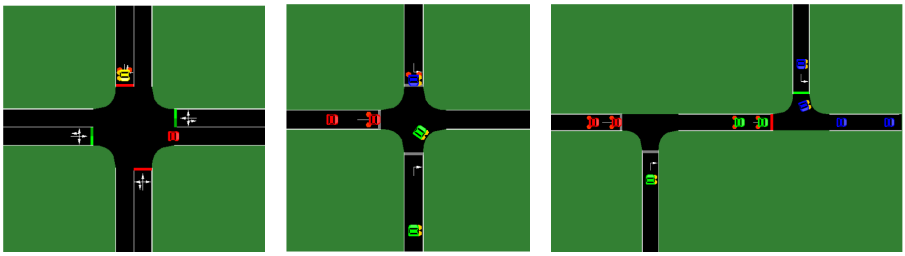
Input: nominal MDP $\mathcal{M}$, expert trajectories $\mathcal{T}$, search space S_M, background knowledge B
Parameter: planning horizon Γ, number of iterations η
Output: hypothesis H

$C \leftarrow \emptyset$
for $i \leftarrow 0$ to η **do**
  for $g \in \mathcal{G}$ **do** ▷ Constraint inference
    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 for
    $\pi(a \mid s, g) \leftarrow \frac{Z_{s,a,g}}{Z_{s,g}}$
  end for
  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 while
  $c^* \leftarrow \arg\max_{s,a} D_{s,a}$ 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 for
return H

4.1 Environments

We conduct experiments in the towers of Hanoi environment (Figure 1) and simulated traffic environments (Figure 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 to illustrate the working of our algorithm. In this environment, there is only one agent which plays the game.



(a) Traffic light controlled intersection

(b) Intersection with priority to the right

(c) Combination of 2a and 2b

Fig. 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.

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 2a), an intersection with unidirectional traffic where the priority is given to cars coming from the right (Fig 2b) and an environment combining both previous scenarios (Fig 2c). 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 [16] 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.

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 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

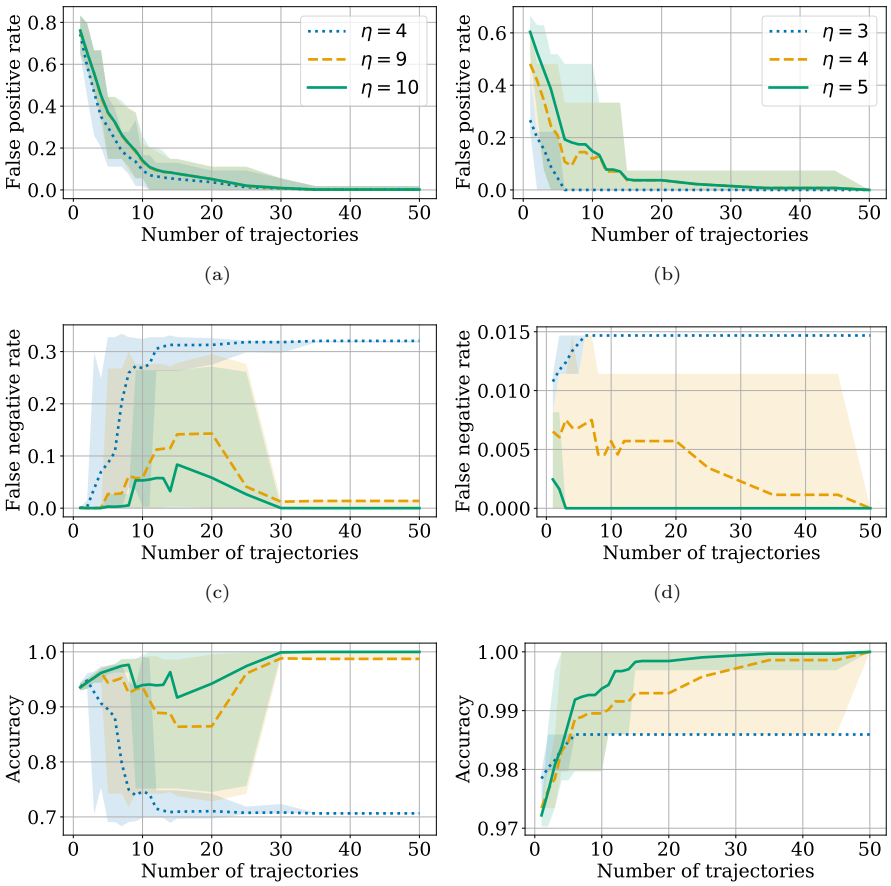


Fig. 3: False positive rate, false negative rate and accuracy of the induced rules for the traffic light controlled intersection environment (3a, 3c, 3e respectively) and for the intersection with priority to the right environment (3b, 3d, 3f respectively) as a function of the number of observed trajectories. Mind the different y-axis scale.

with priority to the right. The last four rules state cars have to stop at a red traffic light.

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

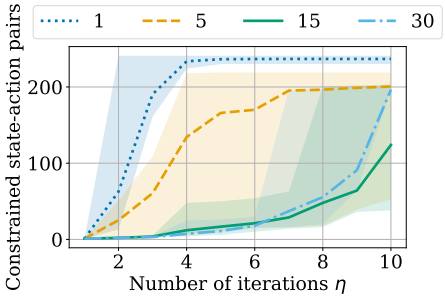


Fig. 4: Number of state-action pairs which are constrained by the inferred rules for increasing values of η. Line styles indicate the number of expert trajectories available. Results are from the traffic light controlled intersection environment.

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 numbers of trajectories (dataset sizes) and iterations η. The results are depicted in Figure 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 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 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

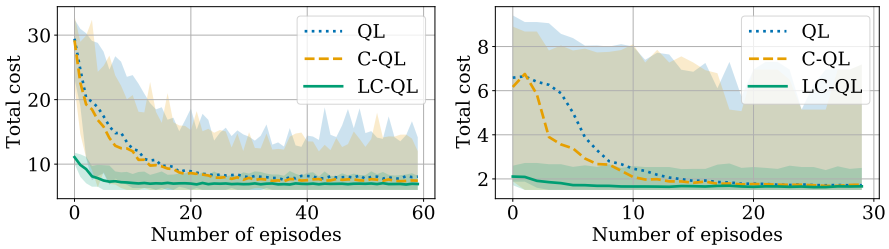


Fig. 5: Total cost received during training using Q-learning in the traffic light controlled junction environment (5a) and the intersection with priority to the right environment (5b).

inferred accordingly the induced rules invalidate less state-action pairs hence a smaller FPR but a higher FNR. It is self-evident the number of constrained state-action pairs increase when the number of iterations increase. Notably, in the traffic light scenario of Figure 3c, 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 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.

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 [17, 18]. The action space is restricted by first translating the induced hypothesis to a set of constraints in state-action space (algorithm 1 line 21). Next, the set of sets of valid actions $\{A_s\}$ is restricted using this set of constraints (algorithm 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.

Table 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**

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 importance of the induction step of algorithm 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 afterwards ($\epsilon = 0$). 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 5 shows the total cost received by an agent trained for a different number of episodes in the intersection with traffic lights environment (5a) and the intersection with priority to the right environment (5b). 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 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 only our method guarantees 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 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 5a and 5b) there is no overall time saving compared to training an agent without prior knowledge.

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

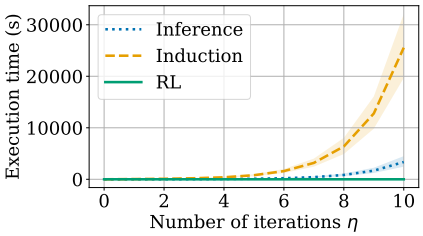


Fig. 6: Execution time of the different steps of our algorithm applied to the traffic light controlled intersection environment.

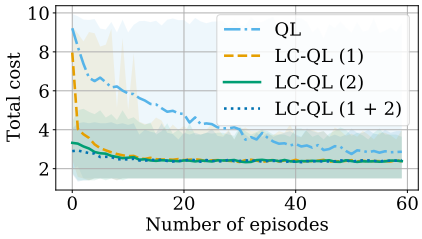


Fig. 7: Transfer learning experiment: total cost received in the combo environment (see figure 2c).

Table 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**

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 2c. 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 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 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 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.

5 Related Work

The first definition of constrained MDP's was provided by Altman [19] 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 research is still ongoing [20, 21]. 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 [22] or by a human guiding the robot [23, 24]. Another study focuses on inferring sequential constraints which capture a task's sequential structure [25]. Previous work [7, 26, 27] 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. [28], 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. [29] 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. [30] 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 [31]. 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 [32]. Other work suggests to decompose the Q-function in a task and a survival component [33]. 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 [34]. 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 [18]. 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. [35] 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 [36], 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 [37] or a policy [38] in the relational domain, given expert demonstrations. However, these approaches build on supervised learning techniques which fail when demonstrations are sub-optimal. 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 [39]. 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 [40] is a subclass of ILP which learns a transition model from traces of a system. The Apperception Engine [41] 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.

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 [42] or PPO [43] 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 ego-centric 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 [44], PPO [43] and Rainbow [42]. 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 [20, 45]: 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" [18] 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 [28] 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 [46] is a potential way to resolve this issue.

7 Declarations

7.1 Funding

This research was partially funded by the Flemish Government (Flanders AI Research Program).

7.2 Conflicts of interest/Competing interests

The authors have no relevant financial or non-financial interests to disclose.

7.3 Ethics approval

Not Applicable.

7.4 Consent to participate

Not Applicable.

7.5 Consent for publication

Not Applicable.

7.6 Availability of data and material

The SUMO traffic simulator is available at https://www.eclipse.org/sumo/.

7.7 Code availability

The full code is available at https://gitlab.ilabt.imec.be/mwbaert/constraint-inference.

7.8 Authors' contributions

All authors contributed to the study conception and design. Material preparation, data collection, coding and analysis were performed by Mattijs Baert. The first draft of the manuscript was written by Mattijs Baert and all authors edited and commented on previous versions of the manuscript. All authors read and approved the final manuscript.

References

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

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

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

[4] Russell, S.: Human Compatible: Artificial Intelligence and the Problem of Control. Penguin, London (2019)

[5] Everitt, T., Hutter, M.: Avoiding wireheading with value reinforcement learning. In: International Conference on Artificial General Intelligence, pp. 12–22 (2016). Springer

[6] Ng, A.Y., Russell, S.J., *et al.*: Algorithms for inverse reinforcement learning. In: Icml, vol. 1, p. 2 (2000)

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

[8] Malik, S., Anwar, U., Aghasi, A., Ahmed, A.: Inverse constrained reinforcement learning. In: International Conference on Machine Learning, pp. 7390–7399 (2021). PMLR

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

[10] Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT press, Cambridge (2018)

[11] Abbeel, P., Ng, A.Y.: Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the Twenty-first International Conference on Machine Learning, p. 1 (2004)

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

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

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

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

[16] Lopez, P.A., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flötteröd, Y.-P., Hilbrich, R., Lücken, L., Rummel, J., Wagner, P., Wießner, E.: Microscopic traffic simulation using sumo. In: The 21st IEEE International

Conference on Intelligent Transportation Systems. IEEE, ??? (2018). https://elib.dlr.de/124092/

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

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

[19] Altman, E.: Constrained Markov Decision Processes: Stochastic Modeling. Routledge, London (1999)

[20] Tessler, C., Mankowitz, D.J., Mannor, S.: Reward constrained policy optimization. arXiv preprint arXiv:1805.11074 (2018)

[21] Wachi, A., Sui, Y.: Safe reinforcement learning in constrained markov decision processes. In: International Conference on Machine Learning, pp. 9797–9806 (2020). PMLR

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

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

[24] Subramani, G., Zinn, M., Gleicher, M.: Inferring geometric constraints in human demonstrations. In: Conference on Robot Learning, pp. 223–236 (2018). PMLR

[25] Pardowitz, M., Zöllner, R., Dillmann, R.: Learning sequential constraints of tasks from user demonstrations. In: Humanoids, pp. 424–429 (2005)

[26] Chou, G., Berenson, D., Ozay, N.: Learning constraints from demonstrations. arXiv preprint arXiv:1812.07084 (2018)

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

[28] Glazier, A., Loreggia, A., Mattei, N., Rahgooy, T., Rossi, F., Venable, K.B.: Making human-like trade-offs in constrained environments by learning from demonstrations. arXiv preprint arXiv:2109.11018 (2021)

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

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

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

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

[33] Van Molle, P., Verbelen, T., Bohez, S., Leroux, S., Simoens, P., Dhoedt, B.: Decoupled learning of environment characteristics for safe exploration. arXiv preprint arXiv:1708.02838 (2017)

[34] Geibel, P.: Reinforcement learning for mdps with constraints. In: European Conference on Machine Learning, pp. 646–653 (2006). Springer

[35] Coppens, Y., Steckelmacher, D., Jonker, C.M., Nowé, A.: Synthesising reinforcement learning policies through set-valued inductive rule learning. In: TAILOR, pp. 163–179 (2020)

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

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

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

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

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

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

[42] Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., Silver, D.: Rainbow: Combining improvements in deep reinforcement learning. In: Thirty-second AAAI Conference on Artificial Intelligence (2018)

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

[44] Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: International Conference on Machine Learning, pp. 1889–1897 (2015). PMLR

[45] Kalweit, G., Huegle, M., Werling, M., Boedecker, J.: Deep constrained q-learning. arXiv preprint arXiv:2003.09398 (2020)

[46] Ziebart, B.D., Bagnell, J.A., Dey, A.K.: Modeling interaction via the principle of maximum causal entropy. In: ICML (2010)

Appendix A 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.

A.1 Towers of Hanoi

A.1.1 Background B

```
1 y(0..2).
2 x(0..2).
3 disk(0..2).
4
5 1{at(D, Y, X) : y(Y), x(X)}1 :- disk(D).
6 on(D1, D2) :- at(D1, X1, Y1), D1!=D2
7       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`.

A.1.2 Language bias M

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

A.1.3 Hypothesis H

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

A.2 Traffic Light Controlled Intersection

A.2.1 Background B

```
1 row(1..5).
2 col(1..5).
3
4 dir(zero).
5 dir(north).
6 dir(east).
7 dir(south).
8 dir(west).
```

```
9
10 1{at(C, R): col(C), row(R)}1.
11 1{tls0; tls1}1.
12 1{go(V) : dir(V)}1.
13
14 onRoad(zero) :- at(X,Y),
15       col(X), row(Y), X=3, Y=3.
16 onRoad(south) :- at(X,Y),
17       col(X), row(Y), X=3, Y<3.
18 onRoad(north) :- at(X,Y),
19       col(X), row(Y), X=3, Y>3.
20 onRoad(east) :- at(X,Y),
21       col(X), row(Y), X>3, Y=3.
22 onRoad(west) :- at(X,Y),
23       col(X), row(Y), X<3, Y=3.
24
25 beforeJunction(south) :- at(X,Y),
26       col(X), row(Y), X=3, Y=2.
27 beforeJunction(north) :- at(X,Y),
28       col(X), row(Y), X=3, Y=4.
29 beforeJunction(west) :- at(X,Y),
30       col(X), row(Y), X=2, Y=3.
31 beforeJunction(east) :- at(X,Y),
32       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 A1.

A.2.2 Language bias M

```
1  #constant(direction, zero).
2  #constant(direction, north).
3  #constant(direction, east).
4  #constant(direction, south).
5  #constant(direction, west).
6
7  #constant(cord, 1).
8  #constant(cord, 2).
9  #constant(cord, 3).
10 #constant(cord, 4).
11 #constant(cord, 5).
12
```

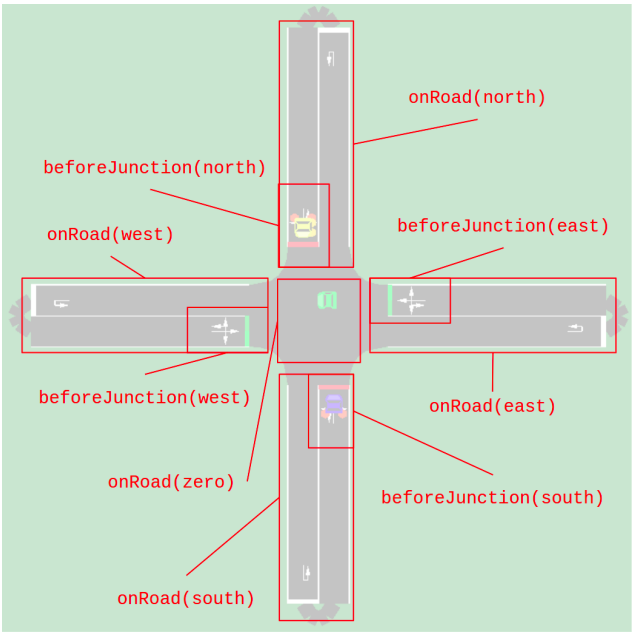


Fig. A1: Meaning of predicates in the traffic light controlled intersection environment.

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

#max_penalty(50).
```

A.2.3 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.
```

A.3 Intersection with Priority to the Right

A.3.1 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.

A.3.2 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).

```

```
13
14 #modeb(1, at(const(cord), const(cord)), (positive)).
15 #modeb(1, onRoad(const(direction))).
16 #modeb(1, go(const(direction))).
17 #modeb(1, carOnTheRight).
18 #modeb(1, onRoad(var(dir))).
19 #modec(1, dir(var(dir))).
20
21 #max_penalty(30).
```

A.3.3 Hypothesis H

```
1 :- not onRoad(V1) : dir(V1).
2 :- go(north); not onRoad(south).
3 :- go(south); not onRoad(north).
4 :- go(east); onRoad(south).
5 :- go(east); onRoad(north).
6 :- go(west).
7 :- carOnTheRight; not go(zero).
```

Appendix B 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.

B.1 Background B

```
1  row(1..7).
2  col(1..7).
3
4  1{at(C, R): col(C), row(R)}1.
5  1{tls0; tls1}1.
6  1{carOnTheRight; noCarOnTheRight}1.
7  1{go(V) : dir(V)}1.
8
9  dir(zero).
10 dir(north).
11 dir(east).
12 dir(south).
13 dir(west).
14
15 onRoad(zero) :- at(X,Y), col(X), row(Y), X=6, Y=4.
16 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.
```