Reinforcement Learning Explanation via Model Transforms

Anonymous Author(s)
Affiliation
Address
email

Abstract

Understanding the emerging behaviors of reinforcement learning agents may be difficult because such agents are often trained using highly complex and expressive models. In recent years, most approaches developed for explaining agent behaviors rely on domain knowledge or on an analysis of the agent's learned policy. For some domains, relevant knowledge may not be available or may be insufficient for producing meaningful explanations. We suggest using formal model abstractions and transforms, previously used mainly for expediting the search for optimal policies, to automatically explain discrepancies that may arise between the behavior of an agent and the behavior that is anticipated by an observer. We formally define this problem of *Reinforcement Learning Policy Explanation* (RLPE), suggest a class of transforms which can be used for explaining emergent behaviors, and suggest methods for searching efficiently for an explanation. We demonstrate the approach on standard benchmarks.

1 Introduction

A major limitation of AI models is the performance-transparency trade-off Rudin (2019). As the inner workings of a model increase in complexity, it becomes more powerful, but the process through which it makes decisions becomes harder to understand. Accordingly, interest in *Explainable Artificial Intelligence* and the development of transparent, interpretable AI models has increased rapidly in recent years Arrieta et al. (2020). This trend is particularly prevalent in *reinforcement learning* (RL), and *deep reinforcement learning* (DRL), where an agent autonomously learns how to operate in its environment. While RL has been successfully applied to solve many challenging tasks, including traffic control Arel et al. (2010), robotic motion planning Kober, Bagnell, and Peters (2013), board games Silver et al. (2016, 2018); Schrittwieser et al. (2020), and more, it is becoming more and more challenging to explain the behavior of RL agents, especially when they do not operate as anticipated. Therefore, in order to allow humans to collaborate with them effectively, it is important to develop methods for reasoning and explaining the agents' behaviors. While there has been a variety of recent works on explainability of DRL (see Puiutta and Veith (2020) for a recent survey), most of these methods do not exploit the full formal model of the environment, typically modeled as a Markov Decision Process (MDP), but instead focus on one element of the model (e.g., the agent's reward function Juozapaitis et al. (2019)) or reasoning about the structure of the underlying network Greydanus et al. (2017). In this work, we suggest a novel approach to explainability that makes use of formal transforms of the underlying environment to automatically generate explanations of emerging behavior. As is common in most RL work, we assume the stochastic environment is modeled as a Markov Decision Process (MDP) Bertsekas (2005). This makes it possible to generate explanations without relying on domain knowledge, and instead using transforms (abstractions) that have been used so far in the literature to expedite RL and planning solutions Bonet and Geffner (2005); Li, Walsh, and Littman (2006); Abel et al. (2018); Yoon et al. (2008). While for planning, the benefit of

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

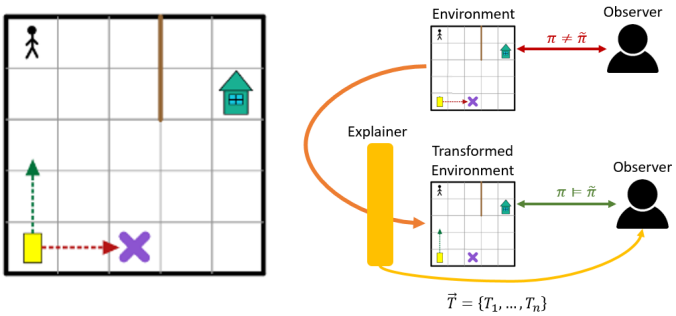

Figure 1: Reinforcement Learning Policy Explanation example (left) and model (right)

such transforms is in highlighting the features of the environment that influence the solution quality and time, we use them to isolate features of the environment that cause an agent to deviate from a behavior that is anticipated by an observer. We consider an explainability setting, which we refer to as *Reinforcement Learning Policy Explanation* (RLPE), as comprised of three entities. The first entity, the *actor*, is an RL agent that seeks to maximize its accumulated reward in the environment. The second entity, the *observer*, expects the actor to behave in some way and to follow a certain policy, which may differ from the one actually adopted by the actor. We refer to this policy as the *anticipated partial policy*, which specifies which actions an observer expects the actor to perform in some set of states. Finally, the *explainer* is an entity that has access to the description of the environment, to the approach (algorithm) used by the actor to compute its policy, to the anticipated partial policy, and to a set of MDP transforms. The explainer seeks a sequence of transforms to apply to the environment so that the actor's policy in the transformed environment aligns with the observer's anticipated policy.[1]

Example 1 *To demonstrate RLPE, consider Figure 1, which depicts a variation of the Taxi domain Dietterich (2000). In this setting, the actor represents a taxi that operates in an environment with a single passenger. The taxi can move in each of the four cardinal directions, pick up, and drop off the passenger. The taxi incurs a small cost for each action it performs in the environment, and gains a high positive reward for dropping off the passenger at her destination. There are walls in the environment that the taxi cannot move through. The observer has some partial view of the environment and actor's model. She knows which actions the taxi can perform and how it can collect rewards. With the information available to her and the possibly incorrect assumptions she makes about the actor's reasoning, the observer anticipates that the taxi will start its behavior by moving towards the passenger (so it can later pick her up and drop her off at her destination). This description of the anticipated behavior over a subset of the reachable states in the environment is the* anticipated partial policy. *The prefix of this policy is depicted by the green arrow in the figure. However, the actual policy adopted by the actor, for which the prefix is represented by the red arrow in Figure 1, is to visit some other location before moving towards the passenger. In order to explain the actor's behavior, the explainer applies different action and state transforms to the environment. The objective is to find a transformed model in which the actor follows the anticipated partial policy. In our example, the explainer first applies an action abstraction that allows the taxi to move through walls and trains the actor in the transformed environment. Since the policy in the transformed model still does not match the anticipated policy, the explainer can infer that the reason for the discrepancy is not the fact that the observer may be unaware of the walls in the environment, and therefore this transform would not represent a meaningful explanation. As a second attempt, the explainer applies an abstraction that relaxes the constraint that a car needs enough fuel to be able to move, and allows the taxi to move regardless of its fuel level. After training, the actor's policy in the transformed environment aligns with the anticipated partial policy. This indicates that the observer may not be aware of the fuel constraint, and therefore does not expect the actor to drive towards the gas station before driving to the passenger. This transform can therefore be used to explain the discrepancy between the anticipated and actual policies.*

[1] In some settings, the observer and explainer may represent the same entity. We use this structure to separate the role of an observer from the attempt to explain the actor's behavior.

Beyond this illustrative example, the ability to understand the "anticipation gap" between an agent and its user is important in many applications. Examples include autonomous driving Kiran et al. (2020), where it might be critical to know why a vehicle deviates from an anticipated course of action. Medical applications, where it is crucial to explain why an AI system recommends one treatment over another Komorowski et al. (2018), and search and rescue missions Kulkarni et al. (2020), where a robot is moving in an unknown environment, far from its operator, and may behave unpredictably. The contributions of this work are threefold. First, we present the first use of formal model transforms and abstractions to produce explanations of RL agent behaviors. Second, we present a formal definition of the Reinforcement Learning Policy Explanation (RLPE) problem and specify classes of state and action abstractions that can be used to produce meaningful explanations. Finally, we implement our approach and provide an empirical evaluation of this approach on a set of standard RL benchmarks.

2 Related Work

A variety of recent studies investigate the use of abstractions to provide meaningful post-hoc explanations for the behavior of RL agents. Most of these methods focus on a particular element of the model and investigate its effect on the agent's behavior. For example, Juozapaitis et al. (2019) decompose the reward function into an aggregation of meaningful reward types, and then classify actions according to their reward type. Lin, Lam, and Fern (2021) use human-designed features to represent action-value functions. The features correspond to meaningful information about the agent's predictions for its future, such as the distance from the goal. Huber et al. (2020) use human user-studies to extract saliency maps for RL agents, which evaluate the relevance of features with regards to mental models, trust, and user satisfaction. Greydanus et al. (2017) and Mundhenk, Chen, and Friedland (2019) use saliency maps to produce visual explanations. Amir and Amir (2018) produce a summary of an agent's behavior by extracting important trajectories from simulated behaviors of the agent. Our approach supports arbitrary transforms and abstraction that can be applied to the environment model without relying on domain specific knowledge or on the particular learning approach of the agent. This variety of transforms that can be used for generating explanation relies on the rich literature of methods suggested for expediting the search for solutions to RL settings Li, Walsh, and Littman (2006); Abel, Hershkowitz, and Littman (2016). Our approach is similar to the work by Sreedharan et al. (2020) which suggests learning partial symbolic models of the environment and task, and identifying missing preconditions to explain the actor's behavior. However this work focuses on deterministic environment where the agent is assumed to be able to compute an optimal plan. In contrast, our work supports arbitrary transforms that are applied to stochastic environments with partially informed RL agents.

3 Background: Reinforcement Learning (RL) and Markov Decision Processes (MDP)

Reinforcement learning (RL) deals with learning policies for sequential decision making of agents that operate in an environment for which the dynamics is not fully known Sutton and Barto (2018) . A common assumption is that the environment can be modelled as a Markov Decision Process (MDP) Bertsekas (2005), which is a widely used formalization for sequential decision-making in stochastic environments. An MDP is typically defined as a tuple $\langle S, s_0, A, R, P, \gamma \rangle$, where S is a finite set of states, $s_0 \in S$ is an initial state, A is a finite set of actions, $R : S \times A \times S \to \mathbb{R}$ is a Markovian and stationary reward function that specifies the reward $r(s, a, s')$ an agent gains from transitioning from state s to s' by the execution of a, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathbb{P}[\mathcal{S}]$ is a transition function denoting a probability distribution $p(s, a, s')$ over next states s' when action a is executed at state s, and $\gamma \in [0, 1]$ is a discount factor, representing the deprecation of agent rewards over time. In this work we use *factored MDPs* Boutilier, Dean, and Hanks (1999), where the set of states is described via a set of random variables $X = X_1, \ldots, X_n$, and where each variable X_i takes on values in some finite domain $Dom(X_i)$. A state is an assignment of a value $X_i \in Dom(X_i)$ for each variable X_i. A solution to an RL problem is either a *stochastic policy*, indicated $\pi : S \to \mathbb{P}[A]$, representing a mapping from states $s \in \mathcal{S}$ to a probability of taking an action a at that state, or a *deterministic policy*, indicated $\pi : S \to A$, mapping from states to a single action. The agent's objective is to find a policy that maximizes the expected accumulated reward.

There are a variety of approaches for solving RL problems Szepesvári (2010); Sutton and Barto (2018); Levine et al. (2020), that can be generally categorized as either *policy gradient methods* (e.g., Stochastic Gradient Ascent) which learn a numerical preference for executing each action, *value-based methods* (e.g., Q-learning), which estimate the values of state-action pairs, and *actor-critic* methods (e.g., A3C Mnih et al. (2016)), which combine the value and policy optimization approaches. Another important distinction exists between model-based methods, where a predictive model is learned, and model-free methods, which learn a control policy directly. In this work, we support this variety by assuming the algorithm that is used by the actor to compute its policy is part of our input. Our requirement is each state can be characterized by a set of possible actions for each state.

4 MDP Transforms

In this work we use formal MDP transforms to explain the behaviors of RL agents. The explanation is generated by searching for a set of transforms to the MDP used to model the underlying environment such that the actor's behavior in the modified model aligns with the observer's expectations. If the transition from the original to the transformed environment is meaningful, the difference between the models can help the observer reason about the actor's behavior, thus representing an explanation. We formally define this process in the following sections, and devote this section to describing various transforms that have been suggested in the literature for expediting the search for optimal MDP policies and that we suggest using for explainability. To account for different modifications that can be applied, we define a *transform* as any mapping $T : \mathcal{M} \mapsto \mathcal{M}$ that can be applied to an MDP to produce another MDP. We use the term "transforms" rather than other common terms used in the literature to avoid restricting our discussion to specific kinds of mappings. In particular, we do not use the terms "abstractions" or "relaxations", since these are terms typically used to represent mappings that expedite and facilitate planning, and we want to account for arbitrary transforms that can be relevant as explanations. Transforms may modify any element of the MDP. We provide some examples of transforms that can be applied, but our framework supports any transform to produce explanations. We start by defining transforms that modify the representation of the MDP's state space.

Definition 1 (State mapping function) *A* state-mapping function $\phi : S \mapsto S^\phi$ maps each state $s \in S$, into a state $s' \in S^\phi$. The *inverse image* $\phi^{-1}(s')$ with $s' \in S^\phi$, is the set of states that map to s' under mapping function ϕ.

When changing the state space of an MDP, we need to account for the induced change to the other elements of the model. For this, we use a state weighing function that distributes the probabilities and rewards of the original MDP between the states in the transformed MDP.

Definition 2 (State weighting function) *Li, Walsh, and Littman (2006) A* state weighting function *of a mapping function ϕ is function $w_\phi : S \mapsto [0, 1]$ where for every $\bar{s} \in S^\phi$, $\sum_{s \in \phi^{-1}(\bar{s})} w(s) = 1$.*

Definition 3 (State-Space Transform) *Li, Walsh, and Littman (2006) Given a state mapping function ϕ and a state abstraction weighting function w_ϕ, a* state space transform *$T_{\phi,w}$ maps an MDP $M = \langle S, s_0, A, R, P, \gamma \rangle$ to $T(M) = \langle \bar{S}, \bar{s_0}, A, \bar{R}, \bar{P}, \gamma \rangle$ where for every action a:*

- $\bar{S} = S^\phi$
- $\bar{s_0} = \phi(s_0)$
- $\bar{R}(\bar{s}, a) = \sum_{s \in w^{-1}(\bar{s})} w(s) R(s, a)$
- $\bar{P}(\bar{s}, a, \bar{s}') = \sum_{s \in \phi^{-1}(\bar{s})} \sum_{s' \in \phi^{-1}(\bar{s}')} w(s) P(s, a, s')$

State-space transforms can, for example, group states. In factored representations, this can be easily implemented by ignoring a subset of the state features. In Example 1, a state-space transform can ignore the fuel level, grouping states that share the same taxi and passenger locations. Another family of transform change the action space.

Definition 4 (Action mapping function) *An* action mapping function *$\psi : A \mapsto A^\psi$ maps of every action in A to an action in A^ψ. The* inverse image *$\psi^{-1}(a')$ for $a' \in A^\psi$, is the set of actions that map to a' under mapping function ψ.*

Various action space transforms have been suggested in the literature for planning with MDPs Mausam and Kolobov (2012); Abel et al. (2020a) and can be used for the RL settings we consider here. For example, in cases where the transition function of the MDP is assumed to be (partially) specified, it is possible to apply the *single-outcome determinization*, where all outcomes of an action are removed (associated with zero probability) except for one. In most practical applications, the most likely or desired outcome is preserved. Another widely used transform is the *all outcome determinization* where the planner can choose a desired outcome Yoon, Fern, and Givan (2007b); Keller and Eyerich (2011a). This transform is typically implemented by creating a separate deterministic action for each possible outcome of the original formulation. In settings where actions are associated with preconditions, it is possible to apply a *precondition relaxation*, by which a subset of the preconditions of an action are ignored Sreedharan et al. (2020). For example, for MDPs represented via a factored state space, each action a is associated with a set $pre_(a)$ specifying the required value of a subset of its random variables. A precondition relaxation transform removes the restriction regarding these variables. Another popular formalism for organizing the action space of an agent are *options*, which represent correlated, long-horizon sequences of actions Sutton, Precup, and Singh (1999); Abel et al. (2020b). Options are defined via three components that indicate at which states the option can be executed, under which condition it terminates, and what is the policy to perform between these conditions. Options are known to expedite learning and support generalization in RL (e.g., Konidaris and Barto (2007); Bagaria et al. (2021a). Such representations can support explanations, in particular for model-based option discovery approaches Bagaria et al. (2021b), where the model itself can be used to reason about the options that are discovered and the agent's behavior.

5 Transforms as Explanations

We view the explainability problem as comprised of three entities: an *actor*, who is an agent operating in the environment, an *observer*, who has some anticipation about the behavior of the actor that may not be met by the actual behavior, and an *explainer*, who wishes to clarify the discrepancy between the anticipated and actual behavior. The input to a *Reinforcement Learning Policy Explanation* (RLPE) problem includes a description of the environment, a description of the behavior (policy) of an RL agent in the environment, a set of possible behaviors an observer expects the actor to follow, and a set of possible transforms that can be applied to the environment.

Definition 5 *A Reinforcement Learning Policy Explanation (RLPE) model is defined by the tuple $R = \langle M, A, \tilde{\pi}, T \rangle$, where:*

- *M is an MDP representing the environment,*
- *$A : \mathcal{M} \rightarrow \Pi$ represents the actor, which is associated with an RL algorithm that it uses to compute a policy $\pi \in \Pi$,*
- *$\tilde{\pi}$ is a set of possible* anticipated partial policies *defined over M an observer expects the actor to follow and,*
- *$T : \mathcal{M} \rightarrow \mathcal{M}$ is a finite set of transforms.*

We assume the actor is a reward-maximizing RL agent. The anticipated behavior of the observer is expressed as a partial policy describing what the observer expects the actor to do in a subset of its reachable states. Clearly, the settings of interest here are those in which the actual policy that the actor follows differs from the anticipated policy. We denote by $\mathcal{T}$ the universal set of transforms. Each transform $T \in \mathcal{T}$ is associated with a mapping function for each of the MDP elements that it alters. We let ϕ_T and ψ_T denote the state and action mapping functions, respectively (when the MDP element is not altered by the transform, the mapping represents the identity function). When a sequence of transforms is applied, we refer to the composite state and action mapping that it induces and define it as follows.

Definition 6 (Composite State and Action Space Function) *Given a sequence $\vec{T} = \langle T_1, \ldots, T_n \rangle$, $T_i \in \mathcal{T}$, the* composite state space function *of $\vec{T}$, is $\phi^{\vec{T}}(s) = \phi_{T_n} \cdot, \ldots, \cdot \phi_{T_1}(s)$. The* composite action space function *is $\psi^{\vec{T}}(s) = \psi_{T_n} \cdot, \ldots, \cdot \psi_{T_1}(s)$.*

The explainer seeks a sequence of transforms that produce an environment where the actor follows a policy that corresponds to the observer's expectations. Formally, we seek a transformed environment where the actor's policy *satisfies* the anticipated policy, i.e., for every state-action pair in the anticipated policy, the corresponding state in the transformed model is mapped to its corresponding action. Given a partial policy π, we let $\mathbb{S}(\pi)$ represent the set of states for which the policy is defined, and define policy satisfaction as follows.

Definition 7 (Policy satisfaction) *Given a partial policy π defined over MDP $M = \langle S, s_0, A, R, P, \gamma \rangle$, a partial policy π' defined over MDP $M' = \langle S', s'_0, A', R', P', \gamma' \rangle$, a state mapping function $\phi : S \mapsto S'$ and an action mapping function $\psi : A \mapsto A'$, π' satisfies π, denoted $\pi' \models \pi$, if for every $s \in \mathbb{S}(\pi)$, $\phi(s) \in \mathbb{S}(\pi')$ and $\psi(\pi(s)) = \pi'(\phi(s))$.*

Clearly, for any two policies there exist state and action mappings that can be applied to cause any policy to satisfy another policy. In order to produce valuable explanations we need meaningful transforms that are applied to the underlying MDP, which change it in a way that highlights the elements of the model that cause unexpected behaviors in the actor's policy. In addition, inspired by the notion of *Minimal Sufficient Explanation* Juozapaitis et al. (2019), we want to minimize the change that is applied to the environment. Intuitively, the more similar the original and transformed MDPs are, the better the explanation. We therefore assume the input to an RLPE problem includes some distance metric $d : \mathcal{M} \times \mathcal{M} \to \mathbb{R}_+$ between a pair of MDPs.There are a variety of metrics for measuring the distance between two MDPs Song et al. (2016); Ammar et al. (2014). A straightforward way to measure the distance between two MDPs is to count the number of transforms that are applied to the original MDP to reach the new MDP, but this definition is meaningful only with *atomic transforms*, that change a single element of the MDP (e.g., changing the transition probabilities of a single action). Another measure is the one suggested by Song et al. Song et al. (2016), where the distance between two MDPs M and $M^{'}$ is calculated by computing the accumulated distance between every state in M and its corresponding state in M'. Wang et. al. Wang, Dong, and Shao (2019) use a representation of an MDP as a bipartite graph of state and action vertices and define state and action distances between two vertices recursively according to the similarity between their neighbors in the graph. They show advantage of this method in quantifying structural similarities compared with the bisimulation approach of Ferns et. al. Ferns, Panangaden, and Precup (2004). The method is designed for measuring similarities within the same MDP, but may be applicable in our context to measure distances between an original and a transformed MDP, as long as a mapping between the states and actions of the models are given. The objective of the explainer is to find a sequence of transforms that yield an MDP M' such that actor's policy in M' satisfies $\tilde{\pi}$. Among the sequences the meet this objective, we are interested in sequences that minimize the distance between the original and the transformed MDP. Formally:

Definition 8 *Given a RLPE model R and a metric function $d : \mathcal{M} \times \mathcal{M} \to \mathbb{R}_+$, a* RLPE problem *seeks a transform sequence $\vec{T} = \langle T_1, \ldots, T_n \rangle$, $T_i \in T$ s.t.*

1. *the policy π' of the actor in $\vec{T}(M)$ satisfies $\tilde{\pi}$, i.e, $\pi' \models \tilde{\pi}$,*
2. *among the sequences that satisfy [1], $\vec{T}$ minimizes the distance $d(M, \vec{T}(M))$.*

6 Finding Explanations

In an RLPE setting, the explainer has access to a set of transforms, but does not know which transform sequence will produce meaningful explanations in a given setting. This means that the explainer may need to consider a large set of possible sequences. Therefore, even though an optimal (i.e., minimum distance) sequence of transforms that satisfies the anticipated policy can be found using an exhaustive search over T, a naive approach is impractical as the number of transform combinations is exponential in $|T|$. To address this computational challenge, we offer three approaches for streamlining the search process. Inspired by the search for an optimal MDP redesign by Keren et al. (2019), a basic approach is a Dijkstra-like search through the space of transform sequences. We assume a successor generator (oracle) is available to provide the MDP that results from a applying each transform. The search graph is constructed in the following way. The root node is the original environment. Each edge (and successor node) appends a single transform to the sequence applied to the parent node, where the edge weight represents the distance between the adjacent MDPs according to the distance measure

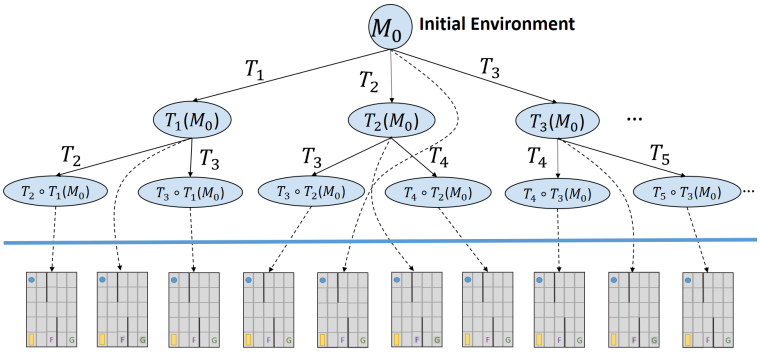

Figure 2: Search in the transform tree.

d. For each explored node we examine whether the actor's policy in the transformed MDP satisfies the anticipated policy. The search continues until such a model is found, or until there are no more nodes to explore. The result is a transform sequence that represents an explanation. This approach is depicted in Figure 2, where the top of the figure depicts the search in the transform space, while the lower part depicts the MDPs corresponding to each transform sequence. This approach is guaranteed to return an optimal (minimum distance) solution under the assumption that the distance is *additive* and *monotonic* w.r.t the transforms in T, in that a transform cannot decrease the distance between resulting MDP and the original one. From a computational perspective, even though in the worst case, this approach covers all the possible sequences, in practice, it may find solutions much quicker. In addition, in cases where the transforms are independent, in that their order of application does not affect the result, it is possible to expedite the search by maintaining a closed list that avoids the re-computation of examined permutations. The depth of the search can also be bounded by a predefined fixed number of transforms. In spite of these computational improvements, the above solution requires recomputing an actor's policy in the transformed environment for each node in the search graph. One way to avoid this is by preserving the agent's policy in the original environment and using it for bootstrapping the re-training of the agent in the transformed environment. Another way to expedite the search is to group together a set of transforms and examine whether applying the set leads to a change in the actor's policy. If this compound transform does not change the actor's policy, we avoid computing the values of the specific transforms. The approach can potentially reduce the computational effort in settings in which aggregation can be done efficiently, such as when transforms have paramterized representations. In our example, if allowing a taxi to move through (all) walls does not change the actor's policy, we avoid computing the value of applying the individual transforms that allow movement through a single wall.

7 Empirical Evaluation

Our empirical evaluation is dedicated to examining the ability to produce meaningful explanations via MDP transforms, and to examine the efficiency of our suggested approaches for finding satisfying explanations. Each RLPE setting includes a description of the underlying environment, the actual policy followed by the actor, and the policy anticipated by the observer. We describe each of these components below, before describing our results.

7.1 Environments

We used three domains: Taxi, Apple Picking, and Frozen Lake. The Taxi domain is an adaption of the domain suggested in Dietterich (2000) with an addition of a fuel constraint, and is described in Example 1. The environment is deterministic and episodic. Episodes end when the passenger arrives at her destination, the taxi runs out of fuel, or the agent completes 60 steps. The Frozen Lake domain Brockman et al. (2016) is a 8×8 gridworld representing a navigation task on an icy lake. There are four actions for moving north, south, east, or west, and the agent's state is its location. The environment description, an example of which is shown in Figure 3, specifies *Start* and *Goal* locations in the corners of the grid, as well as the locations of holes in the ice. The reward is a constant -1 per step until termination, which occurs when the agent reaches the goal (+20 reward) or falls in a

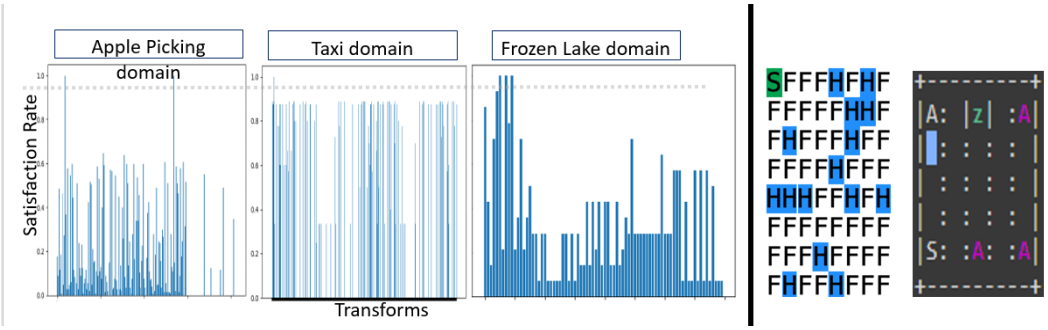

Figure 3: Satisfaction rate per domain (left) and example maps of the Frozen Lake and Apple Picking environments (right).

hole (-20 reward). The Frozen Lake environment is stochastic: when the agent performs the action for direction D, there is a 50% chance of moving in direction D and a 50% chance of slipping on the ice and instead moving in a direction directly clockwise or counterclockwise to D. The Apple Picking environment is a 5×5 gridworld representing a navigation task in an orchard. There are five actions for moving north, south, east, or west, and picking an apple. The environment state consists of the agent location and the status of the apples distributed in the grid in random locations. Figure 3 illustrates an example setting, in which the agent, depicted by a blue square, starts at position 'S'. Each apple is represented as 'A' and the thorny wall as 'z'. The reward for stepping into a thorny wall is -5, and there is a constant -1 per step until termination, which occurs when the agent picks all apples (+20 reward) or the episode terminates. The domain is stochastic: when moving near a thorny wall, the agent succeeds with probability 0.6 and falls into the thorny wall with probability 0.4.

7.2 Setup

Observer: We consider an observer which has partial knowledge about the environment. In the taxi domain, for example, the observer may have access to the map and the reward function, but may be unaware of the fuel constraint, i.e., the need for fuel for movement. For all environments we assume that the observer anticipates that the actor follows an optimal policy in the observer's partial model. We compute the observers anticipated policy using a breadth-first search, using a determinization for stochastic domains. We keep track of multiple optimal paths that may exist, resulting in more than one anticipated policy.

Actor: We use the DQN, SARSA and CEM algorithms, from the keras-rl library[2] to represent the actor. Note that we could use any other algorithm since our framework is agnostic to the method used by the actor. We trained the agents for 600,000 episodes for the Taxi domain, 600,000 episodes for the Frozen Lake domain, and 1,000,000 episodes for the Apple picking domain, each with 60 maximum steps per episode. Our experiments were run on a cluster using a mix of CPUs, with 4 cores using 16GB RAM.

Explainer: For the explainer, we use the following transforms:

- *most likely outcome* which is a special case of single-outcome determinization Yoon, Fern, and Givan (2007a).
- *precondition relaxation* as variant of the usage by Sreedharan et al. (2020) based on STRIPS models Geffner and Bonet (2013).
- *all outcome determinization* suggested by Keller and Eyerich (2011b).

We implemented all three transforms as atomic transforms that each modify a single action. For the most likely outcome, we create one transform for each action. For the all outcome determinization, we create a transform for each action and stochastic outcome pair. For the precondition relaxation transform we created a transform for each action and precondition pair.[3]

[2] *https://github.com/keras-rl/keras-rll*

[3] Further details on the methods and domains are found in the supplementary materials.

	All-outcome		Most likely		Precondition	
	satisfaction	distance	satisfaction	distance	satisfaction	distance
DQN						
Taxi	-	-	-	-	100%	1-4
Frozen Lake	100%	5-8	81%	5-8	-	-
Apple Picking	54%	2-5	56%	2-5	12%	2-4
SARSA						
Taxi	-	-	-	-	100%	1-4
Frozen Lake	0%	-	83%	4-6	-	-
Apple Picking	60%	2-5	59%	2-5	15%	2-4
CEM						
Taxi	-	-	-	-	100%	1-4
Frozen Lake	50%	4	76%	1-4	-	-
Apple Picking	51%	2-5	51%	2-5	8%	2-4

Table 1: Transform satisfaction rate and distance per domain.

We used three methods for searching for explanations, which were detailed in the previous section (for further details see the supplamentary materials). The *BASE* approach is a Dijkstra search, *PRE-TRAIN* is a Dijkstra search in which we bootstrap the learning in the modified environment by using the policy on the original environment. We also prune transforms that do not directly effect the anticipated policy. The third method, *CLUSTER*, computes values of transform clusters.

7.3 Results

To assess the ability to produce explanations using environment transforms, we measured the *satisfaction rate* of each transform in a given environment. This measure is defined as the fraction of the number of states for which the anticipated policy and actor policy agree among all states for which the anticipated policy is defined, i.e., the number of states $s \in \mathbb{S}(\pi)$ for which $\phi(s) \in \mathbb{S}(\pi')$ and $\psi(\pi(s)) = \pi'(\phi(s))$. We also measured the length of the explanation, which we used as our distance measure (that should be minimized). Table 1 presents the average satisfaction rate and distance for each transform and the actor solution method. Settings for which the transforms is irrelevant (e.g., the all outcome determinization is only meaningful in stochastic environment) are indicated by '-'. In each domain we have at least one kind of transform that achieves a satisfaction rate of at least 70% (which means that it produces satisfying explanations for 70% of the examined anticipated partial policies). An interesting result is the fact that the type of transform that produced satisfying explanations varied not only between domains, but between the RL algorithms that were evaluated. In Figure 3 we examine the satisfaction rate achieved per transform in each domain. As can be seen, in each domain there is at least one transform combination that fully satisfies the anticipated policy. For the taxi domain the highest matching rate of 1.0 is reached when the precondition of the fuel for the actions down, right, pickup and drop-off are relaxed. This result aligns with our taxi example where the observer wasn't aware of the fuel limitation. For the apple picking domain, the highest satisfaction rate is achieved by applying the all outcome determinization transform, and the most likely outcome. For the Frozen Lake domain the all outcome determinization achieved the best results. Table 2 compares the performance of the different search methods, showing the ratio of instances that were solved (denoted in the table as "sol") and the average computation time in half hour intervals (denoted in the table as "time") for the instances that were solved by all approaches. The results show that, as expected, the exhaustive BASE method outperforms the two other methods in terms of ratio of instances solved. However, it consumes up to 5 times more computation time. In addition, although the PRE-TRAIN and CLUSTER methods prune transform sequences heuristically, the compromise in terms of solved instances is negligible.

8 Conclusion

We introduced a novel formulation of explainability of RL, and an approach to generating explanations using formal model transforms, which were previously mainly used for facilitating the computation of MDP policies. In addition, we presented approaches for finding a sequence of transforms that represents an explanation to the actor's behavior. Our empirical evaluation on a set of standard RL benchmarks illustrates the ability of formal symbolic MDP transforms to produce explanations that

	BASE		PRE-TRAIN		CLUSTER	
	sol	time	sol	time	sol	time
DQN						
Taxi	99%	5.5h	95%	3h	98%	1h
Frozen Lake	100%	2h	0%	2h	79%	0.5h
Apple Picking	98%	6h	94%	4h	93%	1.5h
SARSA						
Taxi	100%	5h	97%	2.5h	98%	1h
Frozen Lake	85%	3h	0%	3h	81%	0.5h
Apple Picking	99%	6.5h	96%	4h	93%	1.5h
CEM						
Taxi	95%	3h	93%	2.5h	91%	0.75h
Frozen Lake	50%	0.5h	50%	0.5h	74%	0.25h
Apple Picking	92%	4h	88%	3h	78%	1h

Table 2: Comparing method performance.

can be interpreted by a human user, and the efficiency of our approaches for searching through the space of transform sequences.

There are various ways to extend our approach. First, our empirical setup can be enriched to include anticipated policies as they would be expected by human users, and to evaluate the extent by which the generated explanations are meaningful to them. Secondly, while this work uses a restrictive satisfaction relation that required a complete match between the anticipated policy and the actor's behavior, it may be useful to use more flexible qualitative evaluation metrics for satisfaction. Finally, while the focus in this setting has been on single-agent domains, we plan to extend this approach to explain behaviors of agents in multi-agent domains and to add transforms that are relevant to both collaborative and adversarial multi-agent settings.

References

Abel, D.; Arumugam, D.; Lehnert, L.; and Littman, M. 2018. State Abstractions for Lifelong Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning*, 10–19.

Abel, D.; Hershkowitz, D.; and Littman, M. 2016. Near Optimal Behavior via Approximate State Abstraction. In *ICML*.

Abel, D.; Umbanhowar, N.; Khetarpal, K.; Arumugam, D.; Precup, D.; and Littman, M. 2020a. Value Preserving State-Action Abstractions. In *AISTATS*.

Abel, D.; Umbanhowar, N.; Khetarpal, K.; Arumugam, D.; Precup, D.; and Littman, M. 2020b. Value preserving state-action abstractions. In *International Conference on Artificial Intelligence and Statistics*, 1639–1650. PMLR.

Amir, D.; and Amir, O. 2018. HIGHLIGHTS: Summarizing Agent Behaviors to People. In *the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2018)*. Stockholm, Sweden.

Ammar, H. B.; Eaton, E.; Taylor, M. E.; Mocanu, D. C.; Driessens, K.; Weiss, G.; and Tüyls, K. 2014. An automated measure of MDP similarity for transfer in reinforcement learning. *AAAI Workshop - Technical Report*, WS-14-07: 31–37.

Arel, I.; Liu, C.; Urbanik, T.; and Kohls, A. 2010. Reinforcement learning-based multi-agent system for network traffic signal control. *Intelligent Transport Systems, IET*, 128 – 135.

Arrieta, A.; D'iaz-Rodr'iguez, N.; Ser, J.; Bennetot, A.; Tabik, S.; Barbado, A.; Garc'ia, S.; Gil-L'opez, S.; Molina, D.; Benjamins, R.; Chatila, R.; and Herrera, F. 2020. Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI. *ArXiv*, abs/1910.10045.

Bagaria, A.; Senthil, J.; Slivinski, M.; and Konidaris, G. 2021a. Robustly Learning Composable Options in Deep Reinforcement Learning. *IJCAI*.

Bagaria, A.; Senthil, J.; Slivinski, M.; and Konidaris, G. 2021b. Robustly Learning Composable Options in Deep Reinforcement Learning. In Zhou, Z.-H., ed., *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, 2161–2169. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Bertsekas, D. P. 2005. *Dynamic Programming and Optimal Control*, volume I. Belmont, MA, USA: Athena Scientific, 3rd edition.

Bonet, B.; and Geffner, H. 2005. mGPT: A probabilistic planner based on heuristic search. *Journal of Artificial Intelligence Research*, 24: 933–944.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11: 1–94.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym. arXiv:1606.01540.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of artificial intelligence research*, 13: 227–303.

Ferns, N.; Panangaden, P.; and Precup, D. 2004. Metrics for Finite Markov Decision Processes. In *UAI*, volume 4, 162–169.

Geffner, H.; and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers. ISBN 9781608459704.

Greydanus, S.; Koul, A.; Dodge, J.; and Fern, A. 2017. Visualizing and Understanding Atari Agents. *CoRR*, abs/1711.00138.

Huber, T.; Weitz, K.; André, E.; and Amir, O. 2020. Local and Global Explanations of Agent Behavior: Integrating Strategy Summaries with Saliency Maps. arXiv:2005.08874.

Juozapaitis, Z.; Koul, A.; Fern, A.; Erwig, M.; and Doshi-Velez, F. 2019. Explainable Reinforcement Learning via Reward Decomposition. In *in proceedings at the International Joint Conference on Artificial Intelligence. A Workshop on Explainable Artificial Intelligence.*, 111–116.

Keller, T.; and Eyerich, P. 2011a. A polynomial all outcome determinization for probabilistic planning. In *Twenty-First International Conference on Automated Planning and Scheduling*.

Keller, T.; and Eyerich, P. 2011b. A Polynomial All Outcome Determinization for Probabilistic Planning. *Proceedings of the International Conference on Automated Planning and Scheduling*, 21(1): 331–334.

Keren, S.; Pineda, L.; Gal, A.; Karpas, E.; and Zilberstein, S. 2019. Efficient Heuristic Search for Optimal Environment Redesign. *International Conference on Automated Planning and Scheduling*, 246–254.

Kiran, B. R.; Sobh, I.; Talpaert, V.; Mannion, P.; Sallab, A. A. A.; Yogamani, S. K.; and Pérez, P. 2020. Deep Reinforcement Learning for Autonomous Driving: A Survey. *CoRR*, abs/2002.00444.

Kober, J.; Bagnell, J.; and Peters, J. 2013. Reinforcement Learning in Robotics: A Survey. *The International Journal of Robotics Research*, 32: 1238–1274.

Komorowski, M.; Celi, L. A.; Badawi, O.; Gordon, A. C.; and Faisal, A. A. 2018. The Artificial Intelligence Clinician learns optimal treatment strategies for sepsis in intensive care. *Nat Med 24, 1716–1720*.

Konidaris, G. D.; and Barto, A. G. 2007. Building Portable Options: Skill Transfer in Reinforcement Learning. In *IJCAI*, volume 7, 895–900.

Kulkarni, S.; Chaphekar, V.; Chowdhury, M. M. U.; Erden, F.; and Guvenc, I. 2020. UAV Aided Search and Rescue Operation Using Reinforcement Learning. arXiv:2002.08415.

Levine, S.; Kumar, A.; Tucker, G.; and Fu, J. 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

Li, L.; Walsh, T.; and Littman, M. 2006. Towards a Unified Theory of State Abstraction for MDPs. In *ISAIM*.

Lin, Z.; Lam, K.-H.; and Fern, A. 2021. Contrastive Explanations for Reinforcement Learning via Embedded Self Predictions. *ArXiv*, abs/2010.05180.

Mausam; and Kolobov, A. 2012. Planning with markov decision processes: an ai perspective. *Morgan & Claypool Publishers*.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 1928–1937. PMLR.

Mundhenk, T. N.; Chen, B. Y.; and Friedland, G. 2019. Efficient Saliency Maps for Explainable AI. *CoRR*, abs/1911.11293.

Puiutta, E.; and Veith, E. M. 2020. Explainable Reinforcement Learning: A Survey. *ArXiv*, abs/2008.06693.

Rudin, C. 2019. Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. arXiv:1811.10154.

Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; et al. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609.

Silver, D.; Huang, A.; Maddison, C.; Guez, A.; Sifre, L.; Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529: 484–489.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T.; Simonyan, K.; and Hassabis, D. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419): 1140–1144.

Song, J.; Gao, Y.; Wang, H.; and An, B. 2016. Measuring the distance between finite Markov decision processes. *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 468–476.

Sreedharan, S.; Soni, U.; Verma, M.; Srivastava, S.; and Kambhampati, S. 2020. Bridging the Gap: Providing Post-Hoc Symbolic Explanations for Sequential Decision-Making Problems with Black Box Simulators. *CoRR*, abs/2002.01080.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211.

Szepesvári, C. 2010. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1): 1–103.

Wang, H.; Dong, S.; and Shao, L. 2019. Measuring Structural Similarities in Finite MDPs. In *IJCAI*, 3684–3690.

Yoon, S.; Fern, A.; and Givan, R. 2007a. FF-Replan: A Baseline for Probabilistic Planning. 352–.

Yoon, S. W.; Fern, A.; and Givan, R. 2007b. FF-Replan: A Baseline for Probabilistic Planning. In *ICAPS*, volume 7, 352–359.

Yoon, S. W.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *AAAI*, 1010–1016.