Bridging the Gap: Providing Post-Hoc Symbolic Explanations for Sequential Decision-Making Problems with Black Box Simulators

Sarath Sreedharan, Utkash Soni, Mudit Verma ,
Siddharth Srivastava, Subbarao Kambhampati
School of Computing, Informatics, and Decision Systems Engineering,
Arizona State University, Tempe, AZ 85281 USA
{ ssreedh3, usoni1, mverma13, siddharths, rao } @ asu.edu

Abstract

As more and more complex AI systems are introduced into our day-to-day lives, it becomes important that everyday users can work and interact with such systems with relative ease. Orchestrating such interactions require the system to be capable of providing explanations and rationale for its decisions and be able to field queries about alternative decisions. A significant hurdle to allowing for such explanatory dialogue could be the mismatch between the complex representations that the systems use to reason about the task and the terms in which the user may be viewing the task. This paper introduces methods that can be leveraged to provide contrastive explanations in terms of user-specified concepts for deterministic sequential decision-making settings where the system dynamics may be best represented in terms of black box simulators. We do this by assuming that system dynamics can at least be partly captured in terms of symbolic planning models, and we provide explanations in terms of these models. We implement this method using a simulator for a popular Atari game (Montezuma's Revenge) and perform user studies to verify whether people would find explanations generated in this form useful.

1 Introduction

Recent successes in AI have brought the field a lot of attention, and there is a lot of excitement towards deploying AI-based tools for solving various challenges faced in our day to day life. For these systems to be truly effective in the real world, it needs to be capable of working with humans in the loop. This means not just inferring the most optimal action at a given point of time but be able to explain why it chose a particular action intuitively to any users in this loop. Generating such explanations can be particularly challenging when they are trying to explain sequential decisions. Even in deterministic cases, a long chain of actions proposed by the system to achieve a particular goal could be entirely unintelligible for a user owing to complex interrelations between the actions and the users' ignorance about the details of the task. The problem of unintelligibility is further compounded by the fact that the particular systems may not have explicit analytic models of the task specified in interpretable representations. More often than not, these systems may be relying on black-box simulators that operate using high dimensional states for reasoning about the effectiveness of current decisions.

In the end, most naive users will never be able to make sense of the rationale behind each action in terms of system dynamics defined with respect to these high dimensional states (like images of the states or RAM-states for the simulator). Rather the users may be reasoning about the task using some high-level concepts. There is a growing consensus within the explainable AI community that end-user explanations need to be framed in terms of such high-level concepts. Within the context of explaining classifiers, the idea of leveraging such high-level features have been used by works like TCAV [Kim *et al.*, 2018]. Works like [Hayes and Shah, 2017] have also looked at the possibility of leveraging such features to provide policy summaries.

In this work, we aim to use the idea of using learned mappings from high-dimensional simulator states to user-specified concepts to provide post-hoc explanations in goal-directed sequential decision-making problems. Specifically, we will aim to provide contrastive explanations [Miller, 2018] that can be understood as answers to questions of the form *"Why P and not Q?"*, where "P" is the current decisions being proposed by the system (referred to as the *fact* being explained), and "Q" is the infeasible alternative being raised by the user (referred to as the *foil*). We will explain the infeasibility of this foil by assuming that the problem dynamics, at least in states close to the ones appearing in the proposed decision and foil, can be approximated by a symbolic planning model (Section 2), expressed in terms of a set of concepts shared with the user. Given this assumption, we learn missing preconditions of the failing action through interaction with the given black box simulator and the set of learned concept maps from states to concepts (Section 3). Moreover our methods are designed to explicitly allow for the fact that the learned mapping from states to concepts may be noisy (Section 3.2). We validate this approach by instantiating this method on a popular game domain, Montezuma's Revenge [Wikipedia contributors, 2019]. We also performed user studies to verify whether naive users find explanations for contrastive questions as unsatisfied preconditions useful (Section 4).

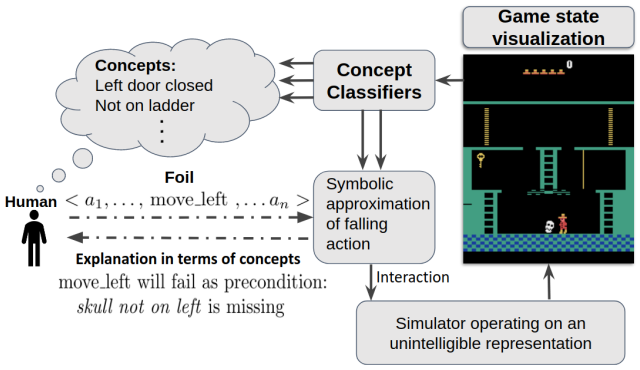

Figure 1: A flow chart for the overall interaction. The explanatory dialogue starts when the user presents the system with a specific alternate plan. The system then uses the simulator to identify specific failure points within the alternate plan. It then tries to identify information about the approximate symbolic model that can capture the dynamics of the failing action by sampling states using the simulator and looking at the concepts predicted by the classifier in those states. Specifically, it tries to identify possible action preconditions that may not be met in the foil.

2 Background

The goal of this work is to address counterfactual queries in terms of high-level concepts with respect to the dynamics of the domain in question. Our focus isn't on how the algorithm came up with the specific decisions but only why this action sequence was chosen instead of an alternative the user may have expected. We, therefore, assume that the specific decision-making algorithm in use will choose the best solution with respect to the current problem and also focus on problems where the original decision-maker and by extension the explanation system has access to some simulator that can be used to predict the outcomes of executing an action on any problem state. Specifically, we assume access to a deterministic simulator of the form $\mathcal{M}_{\text{sim}} = \langle S, A, T \rangle$, where S represents the set of possible world states, A the set of possible actions and T the transition function that specifies the problem dynamics. The transition function is defined as $T : S \times A \rightarrow S \cup \{\perp\}$, where $\perp$ corresponds to a invalid state generated by the execution of an infeasible action. We will look at decision-making problems that are goal-directed in the sense that the decision-making system needs to come up with a sequence of actions $\pi = \langle a_1, .., a_k \rangle$ (henceforth referred to as a plan) that will drive the state of the world to a goal state. In general we will use the tuple $\Pi_{\text{sim}} = \langle I, \mathbb{G}, \mathcal{M}_{\text{sim}} \rangle$ to represent the decision making problem, where I is the initial state and $\mathbb{G}$ is the possible set of goal states. We will be relying on symbolic action models with preconditions (similar to STRIPS style models [Geffner and Bonet, 2013]) as a way to approximate the problem dynamics. In general such models can be represented by the tuple $\Pi_c = \langle F_c, A_c, I_c, G_c \rangle$, where F_c is a set of propositional fluents defining the state space, A_c is the set of actions, I_c the initial state, G_c the goal specification. Each valid problem state in the problem can be uniquely identified by the subset of propositional factors that are true in that state. Each action $a \in A_c$ is further described in terms of the preconditions (specification of states in which a is executable) and the effects of executing the action. In this work, we will mainly focus on models where the preconditions are represented as a conjunction of state factors, i.e., models where $prec_a$ can be expressed as a conjunction of positive literals. If the action is executed in a state where even one of the precondition is missing, then the execution results in the invalid state ($\perp$).

3 Contrastive Explanations

The specific explanatory setting (visualized in Figure 1) we are interested in studying involves a decision-making problem specified by the tuple $\Pi_{sim} = \langle I, \mathbb{G}, \mathcal{M}_{\text{sim}} \rangle$ for which the system identifies a plan π. The user of the system, responds by raising an alternative plan π_f (*the foil*) to be followed instead. Now the system would need to provide an explanation if the foil is invalid and hence can not be followed. We will define an action sequence as invalid if any of its actions could lead to an invalid state ($\perp$) or it doesn't achieve the goal; more formally,

Definition 1. An action sequence π' is said to be invalid for a problem $\Pi_{sim} = \langle I, \mathbb{G}, \mathcal{M}_{\text{sim}} \rangle$, if either of the following conditions are met, i.e

1. If $T(I, \pi') \notin \mathbb{G}$, i.e the sequence doesn't lead to a possible goal state, or
2. If there exists a prefix $\hat{\pi}'$ of π', such that, $T(I, \pi') = \perp$, i.e the execution of it would lead to an invalid state.

To concretize this interaction, consider an instance from the domain of Montezuma's revenge (Figure 1). Let's assume the agent starts from the highest platform, and the goal is to get to the key. The plan π may require the agent to make it's way to the lowest level, jump over the skull and then making its way to the key. Now the user may raise a foil π_f that is quite similar to π except now it involves just moving left (as in trying to move through the skull) instead of jumping over the skull. When the system simulates this alternate sequence, it will find that moving left in this state kills the agent (which in this case is $\perp$ state).

Learning Concept Maps: To describe explanations in this setting, the system needs to have access to a mapping from internal state representation to a set of high-level concepts that are known to the user (for Montezuma this could involve concepts like agent being on ladder, holding onto key, being next to a skull etc.). We will assume each concept corresponds to a propositional fact that the user associates with the task's states and believes that the dynamics of the task are determined by these concepts. This means that as per the user for each given state, a subset of these concepts may be either present or absent. One way to utilize such concepts for explanations would be to learn a binary classifier for each such concept that identifies whether the fact is present or absent in a given state. Such high-level concepts could be specified by the end-user by identifying positive and negative example states for each concept. These examples could be then be used to learn the required classifiers by using algorithms best suited for the internal simulator state representation. Note that this implicitly assumes that the explanatory system has some

method of exposing simulator states to the user. A common way to satisfy this requirement would be by having access to visual representations for the states. The simulator state itself doesn't need to be an image as long as we have a way to visualize it (as in the case of Atari games with their RAM based state representation).

Once learned, such classifiers provide us with a way of converting simulator states to a factored representation. Such techniques have not only been used in explanation (c.f [Kim *et al.*, 2018; Hayes and Shah, 2017]) but also in works that have looked at learning high-level representations for continuous state-space problems [Konidaris *et al.*, 2018].

Let $\mathbb{C}$ be the set of classifiers corresponding to the high-level concepts. For state $s \in S$, we will overload the notation $\mathbb{C}$ and specify the concepts that are true as $\mathbb{C}(s)$, i.e., $\mathbb{C}(s) = \{C_i | C_i \in \mathbb{C} \wedge C_i(s) = 1\}$

Explanation Using Concepts: Now to explain the infeasibility of the foil, we only need to explain the first failing action, i.e., the last action in the shortest prefix that would lead to an invalid state (which in our example is the move-left action in the state presented in Figure 1). One possibility would be to point out to the user that the action failure is due to one of the concepts missing in the state (along with communicating the missing concepts). While this is a necessarily true statement given our structural assumption, there are a few issues with adapting this method. To start with, there could be a large number of concepts that are absent at any given state, and many of them could be irrelevant to the failed action. For example, in the Figure 1, the fact that the agent doesn't have key or is not on a rope or a ladder wouldn't have anything to do with the agent not being able to move left. Moreover, given the assumption that the user knows these concepts, they could have come to the conclusion that failure of the action is due to one of the missing concepts on their own by viewing the image.

A more specific explanation could be one that points to one specific concept absent from the state that is a precondition for the action. In our example, a possible explanation could be to inform the user that the agent can only perform move-left action in states for which the concept agent-not-next-to-the-skull-on-left is true; however the concept is false in the given state. Note that this formulation is enough to capture both conditions for foil failure by appending an additional goal action at the end of each sequence, such that the action causes the state to transition to an end state and it fails on all simulator states except the ones in $\mathbb{G}$. Now we can explain the action failure in terms of what preconditions were not met in the failure state, or more formally:

Definition 2. For a failing action a_i for the foil $\pi_f = \langle a_1, .., a_i, .., a_n \rangle$, $C_i \in \mathbb{C}$ is considered an explanation for failure if $C_i \in prec_{a_i} \setminus \mathbb{C}(s_i)$, where s_i is the state where a_i is meant to be executed (i.e $s_i = T(I, \langle a_1, .., a_{i-1} \rangle)$).

One way to get such an explanation could be to learn a full approximate symbolic model for the task. Unfortunately this could be extremely expensive, particularly if the task has a huge action space. The easier option here would be to focus on learning information about the failing action. Moreover, to explain action failure, we neither need to learn the effects nor all of the preconditions of the given action, rather we just need to find a single precondition that was missing from the failing state.

Representational Assumptions: Before we delve further into the specifics of our method, a quick note on some of the central representational assumptions being made in this work. The obvious one being that it is possible to approximate the applicability of actions in terms of high-level concepts. Apart from the intuitive appeal of such models (many of these models have their origin in models from folk psychology), these representation schemes have been widely used to model real-world sequential decision-making problems from a variety of domains and have a clear real-world utility [Benton *et al.*, 2019]. We agree that there may be problems where it may not be directly applicable, but we believe this is a sound initial step and applicable to many domains where currently RL based decision-making systems are being successfully used, including robotics and games. Apart from this basic assumption, we make two additional representational assumptions

1. The given set of concepts $\mathbb{C}$ suffice to represent the action preconditions
2. The precondition can be expressed as a conjunction of positive concepts

Note that neither assumption restricts the applicability of the methods discussed here. We can easily detect cases where first assumption is not met by a small extension of our methods (further discussed in Section 6). For the second assumption, our framework can still cover cases where the action may require non-conjunctive preconditions. To see why, consider a case where the precondition of action a is expressed as an arbitrary propositional formula $\phi(\mathbb{C})$. In this case, we can express it in its conjunctive normal form $\phi'(\mathbb{C})$. Now each clause in this CNF can be treated as a new compound positive concept. Thus we can cover such arbitrary propositional formulas by expanding our concept list with compound concepts (including negations and disjuncts) whose value is determined from the classifiers for the corresponding atomic concepts.

3.1 Identifying Preconditions through Trials

Now to identify such preconditions, we will rely on the simple intuition that while executability of an action a in the state s_j with a concept C_i doesn't necessarily establish that C_i is a precondition, we can guarantee that any concepts false in that state can not be a precondition of the original action. This is obvious from the semantics of the models we are considering, where an action a is executable in state s_j only if $prec_a \subseteq \mathbb{C}(s_j)$. This is a common line of reasoning exploited by many of the model learning methods (c.f [Carbonell and Gil, 1990; Stern and Juba, 2017]).

This leads us to Algorithm 1 for identifying the precondition. The algorithm takes as input the failed action (a_{fail}), the state at which it was supposed to be executed per the foil (s_{fail}), a sampler over the states of the problem (Sampler), the possible concepts ($\mathbb{C}$) and an upper bound on the number of samples to be explored (ℓ). Any locality assumptions we want to enforce can be baked into the sampler and a simple way to generate such samplers could be by leveraging

Algorithm 1 Algorithm for Finding Missing Precondition

procedure BASIC-SEARCH
Input: $s_{\text{fail}}, a_{\text{fail}}, \text{Sampler}, \mathcal{M}_{\text{sim}}, \mathbb{C}, \ell$
Output: Missing precondition C_{prec}
Procedure:
    poss_precondition_set = $\mathbb{C} \setminus \mathbb{C}(s_{\text{fail}})$
    sample_count = 0
    while sample_count $< \ell$ **do**
        $s \sim$ Sampler
        if $T(s, a_{\text{fail}}) \neq \perp$ **then**
            poss_prec_set = poss_prec_set $\cap$ $\mathbb{C}(s)$
        if |poss_prec_set| == 1 **then return** $C_i \in$poss_prec_set
        sample_count += 1
    return Any concept $C_i \in$poss_prec_set

random walk either from initial states or states from the foil or the proposed plan. The algorithm starts by initializing the set of possible missing preconditions to all concepts missing from the failure state (for the given example, this could include concepts like on_rope, on_ladder, on_highest_platform, not_next_to_skull_on_left). Then we start exploring other possible problem states, and we use each new state where the action is executable to reduce our possible precondition list further (since any concept not part of this state can't be a possible precondition). For example, one might find that we can execute move_left on the highest platform, this could lead to the elimination of concepts like on_rope, on_ladder etc. If we know that all concepts required to characterize the preconditions are given upfront, then whenever the set of remaining concepts drops to one, we can exit the loop and return the remaining concept. This is because, as per our setting, there must be at least one missing precondition in the original state. Unfortunately, if there are multiple preconditions that are missing in the original failure state or we are not sure whether the concept set is complete, then we will have to wait until we exhaust the sampling budget. In the limit, the algorithm is guaranteed to identify all the missing preconditions. If at the end of the search, our possible set is empty, for cases where the classifier is perfect, this means that the given concept list is incomplete, and we require more user-specified concepts.

Confidence over explanations: Even with a large sampling budget, we may still want to establish some level of confidence in the solutions identified. We can quantify this by leveraging the intuition that among all the states where the action is executable, all non-precondition concepts should be randomly distributed. This means that every time we sample a state where an action is executable, and the concept is present, we should be able to increase our confidence in the concept being a precondition. We can operationalize this intuition by making the following assumptions (1) the distribution of concepts over the state space is independent of each other and (2) the distribution of all non-precondition primitive concepts is the same as their overall distribution across the problem states (which can be empirically estimated). Specifically, we will assume the probabilistic relationship between the random variables is as captured by Figure 2 (A). Where $O^s_{a\_e}$ corresponds to the fact that current state s allows for the execution of the current action a, $C_i \in prec_a$ is the fact that the concept is in a precondition of a and $O^s_{C_i}$ is the fact that we observe the concept C_i for the state s. Note that here we assume the mapping from state to concepts are accurate and thus we make no distinction here between classifier returning a concept and the concept being part of the state.

For a single observation of a state where C_i is true and a is executable (denoted as $O^s_{C_i} \wedge O^s_{a\_e}$), we can calculate the posterior probability as

$$P(C_i \notin prec_a | O^s_{C_i} \wedge O^s_{a\_e}) = \frac{P(O^s_{C_i} | C_i \notin prec_a \wedge O^s_{a\_e}) * P(C_i \notin prec_a | O^s_{a\_e})}{P(O^s_{C_i} | O^s_{a\_e})}$$

Given $C_i \notin prec_a$ is independent of $O^s_{a\_e}$ and expanding the denominator we get

$$= \frac{P(O^s_{C_i} | C_i \notin prec_a \wedge O^s_{a\_e}) * P(C_i \notin prec_a)}{P(O^s_{C_i} | C_i \notin prec_a \wedge O^s_{a\_e}) * P(C_i \notin prec_a) + P(O^s_{C_i} | C_i \in prec_a \wedge O^s_{a\_e}) * P(C_i \in prec_a)} \quad (1)$$

From our assumption, we know $P(O^s_{C_i} | C_i \notin prec_a \wedge O^s_{a\_e})$ is same as the distribution C_i over the problem states (p_{C_i}) and $P(O^s_{C_i} | C_i \in prec_a \wedge O^s_{a_e})$ must be one. We can substitute these values in directly calculate the posterior and then calculate $P(C_i \in prec_a | O^s_{C_i} \wedge O^s_{a\_e})$ as $1 - P(C_i \notin prec_a | O^s_{C_i} \wedge O^s_{a\_e})$. We can similarly calculate the posterior given all the observations. We can either return this probability for all the observation as the probability of the explanation being true or even use it as a way to exit the loop once it crosses a certain limit. Probabilities for compound concepts can be calculated from the individual atomic concepts.

3.2 Using Noisy Concept Classifiers

Given how unlikely it is that we have access to a perfect classifier, a more practical assumption to adopt could be that we have access to a noisy classifier, but we also have access to a probabilistic model for its prediction. That is, we have access to a function $P_{\mathbb{C}} : \mathbb{C} \rightarrow [0, 1]$ that gives the probability that the concept predicted by the classifier is actually associated with the state. Such probability functions could be learned from the test set used for learning the classifier. A direct consequence of such a probabilistic mapping is the fact that you can no longer use a single failure (i.e., execution of an action in a state where the concept is absent) as evidence for discarding the concept. Though we can still use it as evidence to update the probability of a given concept being a precondition. We can remove a particular precondition from consideration once the probability of it not being a precondition crosses a specified threshold.

To see how we can incorporate these probability measures into our search, consider the updated relationships presented in Figure 2 (B). Note that in previous sections, we made no distinction between the concept being part of the state and actually observing the concept. Now we will differentiate between the classifier saying that a concept is present ($O^s_{C_i}$) from the fact that the concept is part of the state ($C_i \in S$). We will assume that the probability of the classifier returning

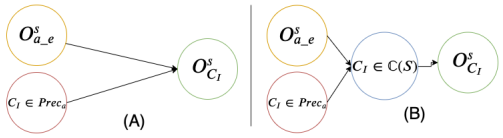

Figure 2: A simplified probabilistic graphical model, Subfigure (A) assumes classifiers to be completely correct, (B) takes care of, when the classifier output may carry confidence measures.

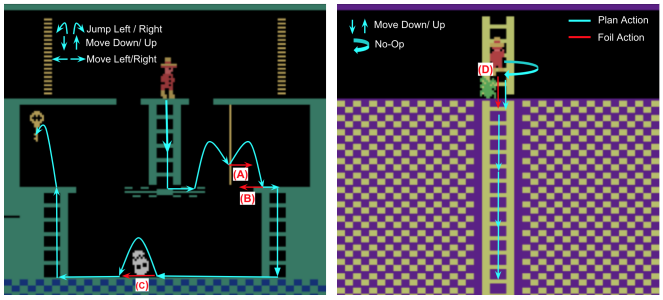

Figure 3: Left Image shows foils for screen 1, (A) Move right instead of Jump Right (B) Go left over the edge instead of using ladder (C) Go left instead of jumping over the skull. Right Image shows foil for screen 4, (D) Move Down instead of waiting.

label and concept being present is given by the probabilistic confidence provided by the classifier. Of course, this still assumes the classifier's model of prediction is accurate. However, since it is the only measure we have access to we will treat it as being correct.

Now instead of using a single instance of absence of a concept in a state where the action is executable as the proof that it is not a precondition, we can use each instance of an absence of concept per the classifier as an observation to update the probability of the concept not being a precondition.

$$P(C_i \notin prec_a | O_{\neg C_i} \wedge O^s_{a\_e}) = \frac{P(O_{\neg C_i} | C_i \notin prec_a \wedge O^s_{a\_e}) * P(C_i \notin prec_a | O^s_{a\_e})}{P(O_{\neg C_i} | O^s_{a\_e})} \quad (2)$$

We can expand $P(O_{\neg C_i} | C_i \notin prec_a \wedge O^s_{a\_e})$ as follows

$$P(O_{\neg C_i} | C_i \notin prec_a \wedge O^s_{a\_e}) = P(O_{\neg C_i} | C \in \mathbb{C}(s)) * P(C \in \mathbb{C}(s) | C_i \notin prec_a \wedge O^s_{a\_e}) + P(O_{\neg C_i} | C \notin \mathbb{C}(s)) * P(C \notin \mathbb{C}(s) | C_i \notin prec_a \wedge O^s_{a\_e})$$

Where as defined earlier $P(C \notin \mathbb{C}(s) | C_i \notin prec_a \wedge O^s_{a\_e})$ and $P(C \in \mathbb{C}(s) | C_i \notin prec_a \wedge O^s_{a\_e})$ would correspond to p_{C_i} and $(1 - p_{C_i})$. Note that unlike the earlier setting, we can't just rely on the classifiers outputs directly to calculate these values from samples. This means we either need to again use the confidence probabilities to estimate the values from a given set of state samples or use the ground truth labels which would anyway be needed to train the classifiers. Similar to Equation 1, we can calculate $P(O_{\neg C_i} | O^s_{a\_e})$ by marginalizing it over $C \in \text{prec}(a)$ and now also over $C \in \mathbb{C}(s)$. We can also extend Equation 1 to incorporate these probabilities.

4 Evaluation

For validating the soundness of the methods discussed before, we tested the approach on the open-AI gym implementation of Montezuma's Revenge [Brockman *et al.*, 2016]. We used the deterministic version of the game with the RAM-based state representation. We considered executing an action in the simulator that leads to the agent's death (falling down a ledge, running into an enemy) or a non NOOP action that doesn't alter the agent position (trying to move left on a ladder) as action failures. We selected four possible foils for the game (illustrated in Figure 3), three from screen 1 and one from screen 4. The base plan in screen 1 involves the agent reaching the key, while screen 4 required the agent to cross the level. For each screen, we came up with ten base concepts and created ten more concepts by considering the negations of them. All state samples (used to generate the samples for the classifier and the algorithm) were created by randomly selecting one of the states from the original plan and then performing random walks from the selected state.

Concept Learning and Precondition Identification: For training each concept classifier, we used game-specific logic and RAM byte values to identify each positive instance and then randomly selected a set of negative examples. We used around 600 positive examples (except for concepts $skull\_on\_right$ and $skull\_on\_left$ in Screen 1 which had 563 and 546 examples respectively) and double the count of negative examples for each concept. These RAM state examples were fed to a binary AdaBoost Classifier [Freund *et al.*, 1999] (Scikit-learn implementation [Pedregosa *et al.*, 2011] version 0.22.1, default parameters), with 70% of samples used as train set and rest as the test set, for each concept. We used a threshold of 0.55 on classifiers for concepts of screen 1, such that a given state has a given concept when the classifier probability is greater than 0.55, to reduce false positives. Finally, we obtained a test accuracy range of 98.57% to 100%, with an average of 99.72% overall concepts of both the levels. All the samples used for the classifier were collected from 5000 sampling episodes for screen 1 and 4000 sampling episodes for screen 4. The concept distribution was generated using 4000 episodes, and the distribution of concepts ranged from 0.0005 to 0.206. Since, for some of the less accurate models, we did observe false negatives resulting in the elimination of the accurate preconditions and empty possible precondition set. So we made use of the probabilistic version of the search (described in Section 3.2) with observation probabilities calculated from the test set. We applied a concept cutoff probability of 0.01, and in all cases, the precondition set reduced to one element in under the 500 step sampling budget (with mean probability of 0.5044 for foils A, B & C. Foil D, in screen 4, gave a confidence value of 0.8604). The ones in screen 1 had lower probabilities since they were based on more common concepts and thus their presence in the executable states is not strong evidence for them being a precondition.

User Study: With the basic explanation generation method in place, we were interested in finding out if actual users would find such an explanation helpful. Would people without a background in AI accept explanations where the failure

of an action is presented in terms of an unmet precondition? Specifically, the hypothesis we wanted to test was

Hypothesis 1*: Would users find missing precondition information a useful explanation for action failures.*

To evaluate this, we performed a user study with the four foils we tested on along with the generated explanation. In the study, each participant was presented with a subset of the concept we used for the study (around five), then shown the plan and a possible foil. Now the participant was shown two possible explanations for the foil. One that showed the state at which the foil failed along with the information that the action cannot be executed in that state and the other reported that the action failed because of a specific concept was missing (the order of the options was randomized). We further provided the user with a five-point Likert scale on whether they felt the explanation they selected was complete and a free text field to provide any additional information they felt would be useful. In total we collected data from 20 participants, where 7 were women, the average age was 25, and 10 people had taken an AI class. We found that 19 out of the 20 participants selected precondition based explanation as a choice. On the question of whether the explanation was complete, we had an average score of 3.35 out of 5 on the Likert scale (1 being not at all complete and 5 being fully complete). The results seem to suggest that information about missing precondition are useful explanations though may not be complete. While not a lot of participants provided information on what other information would have been useful, the few examples we had generally pointed to providing more information about the model (for example, information like what actions would have been possible in the failed state). For completeness we may need to also inform the user about other parts of the model (that may still be relevant to the current plan/foil) and this is something we plan to investigate in future work.

5 Related Work

There is an increasing number of works investigating the use of high-level concepts to provide meaningful post-hoc explanations to the end-users. As mentioned before, the representative works in this direction include TCAV [Kim *et al.*, 2018] and its various offshoots. Works like [Luss *et al.*, 2019] have also looked at using such high-level concepts to come up with related positive and negative examples for a given contrastive query in regards to a specific classification result.

Also, a work that closely relates to the problem discussed in this paper is the work presented in [Hayes and Shah, 2017]. The work specifically looks at the use of high-level concepts in providing policy summaries. In their approach, an answer to queries of type "Why not action a_i?", takes the form of a logical characterization of states where the policy chooses to use action 'a_i'. This means they only need to focus on a much smaller number of states and can also perform prior policy minimization in terms of the high level concepts (which can't be performed in our case as the foil could be extremely different from the original plan). On the other hand, while characterizing alternative policy states may require the use of complex logical formula, our goal of identifying precondition means we are working with a much simpler structure. This also allows us to provide more robust probabilistic guarantees on the output produced by the algorithm. Another related work is the approach studied in [Madumal *et al.*, 2019]. Here they are also trying to characterize dynamics in terms of high-level concepts. Though in their example, they assume that the full structural relationship between the various variables is provided upfront. Also, as mentioned, our work focuses on providing explanations that try to explain the choice of current decisions over alternatives in terms of system dynamics. Works within explainable planning have referred to such explanations [Langley, 2019] as preferential explanations instead of process explanations that try to explain why the given algorithm may have come up with the specific decisions.

6 Conclusion

The paper introduces methods that can be leveraged to generate explanations for specific decisions by allowing users to query the system about alternative plans. The approach aims to provide a rationale for the infeasibility of such alternative plans by providing explanations in terms of user-specified high-level concepts. We implemented the system for the Atari game Montezuma's Revenge and evaluated the explanations generated by our approach by running user studies. While contrastive explanations are answers to questions of the form "Why P and not Q?", we have mostly focused on refuting the foil (the "not Q?" part). This is because, in the presence of a simulator, it is easier to show why the plan is valid by simulating the plan and presenting the state trace. We can further augment such traces with the various concepts that are valid at each step of the trace.

We view the approaches discussed in the paper as the first step towards designing more general symbolic explanatory methods for sequential decision-making problems. In addition to relaxing assumptions about the dynamics of the problem (stochasticity, observability etc.), there are more fundamental extensions to be considered. First would be to allow for refutation of foils that are valid but sub-optimal. In such a case, we can use the concepts to characterize the conditional cost of executing an action, as $C(s, a)$ and use an approach similar to one discussed here to find the minimum set of concepts needed to characterize all the high-cost state-action pairs in the given foil. Second, the work currently assumes access to foils that are fully specified plans, which may not always be available. In such cases, we might need to build a planning model in terms of the given concepts, that can approximate the system dynamics for a subset of actions that can be used to complete the foil. Once we have access to such models, we can directly use methods like [Sreedharan *et al.*, 2019] to provide explanations. Lastly, we can extend our work to deal with cases where concepts required for the failing action may be missing. If we know that the concept set is possibly incomplete, we can force the algorithm to finish the sampling budget which may return an empty set of concepts. Then, we can try to find pairs of states, where both have the same concepts yet the action fails in one and executes in the other, to query the user for additional concepts that can differentiate between them. These concepts can then be added to the current vocabulary set.

Acknowledgments

Kambhampati's research is supported in part by ONR grants N00014-16-1-2892, N00014-18-1-2442, N00014-18-1-2840, N00014-9-1-2119, AFOSR grant FA9550-18-1-0067, DARPA SAIL-ON grant W911NF-19-2-0006, NSF grants 1936997 (C-ACCEL), 1844325, NASA grant NNX17AD06G, and a JP Morgan AI Faculty Research grant.

References

[Benton *et al.*, 2019] J. Benton, Nir Lipovetzky, Eva Onaindia, David E. Smith, and Siddharth Srivastava, editors. *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, ICAPS 2018, Berkeley, CA, USA, July 11-15, 2019*. AAAI Press, 2019.

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

[Carbonell and Gil, 1990] Jaime G Carbonell and Yolanda Gil. Learning by experimentation: The operator refinement method. In *Machine learning*, pages 191–213. Elsevier, 1990.

[Freund *et al.*, 1999] Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[Geffner and Bonet, 2013] Hector Geffner and Blai Bonet. A concise introduction to models and methods for automated planning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 8(1):1–141, 2013.

[Hayes and Shah, 2017] Bradley Hayes and Julie A Shah. Improving robot controller transparency through autonomous policy explanation. In *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 303–312. IEEE, 2017.

[Kim *et al.*, 2018] B. Kim, Wattenberg M., J. Gilmer, Cai C., Wexler J., , F. Viegas, and R. Sayres. Interpretability Beyond Feature Attribution: Quantitative Testing with Concept Activation Vectors (TCAV) . *ICML*, 2018.

[Konidaris *et al.*, 2018] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018.

[Langley, 2019] Pat Langley. Varieties of Explainable Agency. In *XAIP, ICAPS*, 2019.

[Luss *et al.*, 2019] Ronny Luss, Pin-Yu Chen, Amit Dhurandhar, Prasanna Sattigeri, Karthikeyan Shanmugam, and Chun-Chen Tu. Generating contrastive explanations with monotonic attribute functions, 2019.

[Madumal *et al.*, 2019] Prashan Madumal, Tim Miller, Liz Sonenberg, and Frank Vetere. Explainable reinforcement learning through a causal lens, 2019.

[Miller, 2018] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 2018.

[Pedregosa *et al.*, 2011] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Sreedharan *et al.*, 2019] Sarath Sreedharan, Siddharth Srivastava, David Smith, and Subbarao Kambhampati. Why cant you do that hal? explaining unsolvability of planning tasks. In *Proc. IJCAI*, 2019.

[Stern and Juba, 2017] Roni Stern and Brendan Juba. Efficient, safe, and probably approximately complete learning of action models. *arXiv preprint arXiv:1705.08961*, 2017.

[Wikipedia contributors, 2019] Wikipedia contributors. Montezuma's revenge (video game) — Wikipedia, the free encyclopedia, 2019. [Online; accessed 14-January-2020].