In ICAPS 2021 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS 2021)
Guangzhou, China (Virtual)

# Learning User-Interpretable Descriptions of Black-Box AI System Capabilities

**Pulkit Verma, Shashank Rao Marpally,** and **Siddharth Srivastava**
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University, Tempe, AZ 85281, USA
{verma.pulkit, smarpall, siddharths}@asu.edu

## Abstract

Several approaches have been developed to answer specific questions that a user may have about an AI system that can plan and act. However, the problems of identifying which questions to ask and that of computing a user-interpretable symbolic description of the overall capabilities of the system have remained largely unaddressed. This paper presents an approach for addressing these problems by learning user-interpretable symbolic descriptions of the limits and capabilities of a black-box AI system using low-level simulators. It uses a hierarchical active querying paradigm to generate questions and to learn a user-interpretable model of the AI system based on its responses. In contrast to prior work, we consider settings where imprecision of the user's conceptual vocabulary precludes a direct expression of the agent's capabilities. Furthermore, our approach does not require assumptions about the internal design of the target AI system or about the methods that it may use to compute or learn task solutions. Empirical evaluation on several game-based simulator domains shows that this approach can efficiently learn symbolic models of AI systems that use a deterministic black-box policy in fully observable scenarios.

## 1 Introduction

AI systems are rapidly developing to an extent where they can be expected to be used by non-experts who may not understand how they work or what they can and cannot do. Ongoing research on the topic focuses on the significant problem of how to answer such a user's questions about the system's behavior (Anjomshoae et al. 2019; Barredo Arrieta et al. 2020; Chakraborti et al. 2017; Dhurandhar et al. 2018). However, most non-experts hesitate to ask questions about new tools and quite often do not know which questions to ask in order to assess the safe limits and capabilities of an AI system. This problem is aggravated in situations where an AI system can carry out planning, or sequential decision making, and the user's conceptual vocabulary may not be expressive enough to express rich simulator-based models of AI systems and the solution policies they may use. Lack of understanding of the limits of an imperfect system can result in unproductive usage or, in the worst-case, serious accidents (Randazzo 2018). This in turn limits the adoption and productivity of the AI systems.

This work presents a new approach for learning user-understandable expressions of the capabilities of a black-box AI system. This target AI system may use arbitrary internal models, representations and processes for computing solutions to user-assigned tasks. This paradigm captures a wide variety of AI agents, including ATARI-game playing agents that may use a deep-learned $Q$ function for selecting actions, as well as agents that carry out variants of automated planning.

Consider the case of an autonomous car. A non-expert may not even know that they need to ask about why the car is driving so close to the curb before it is too late. While the developer of the system may be willing to share precise details about the car's steering policies (perhaps along with a simulator), non-experts would be hard-pressed to know in advance that that is an aspect they should be worrying about. Deploying such AI systems would be easier if a user could just indicate terms that they understand (e.g., qualitative notions of distance, collisions, and accelerations) and get agent models of the form "if you ask the car to park when the curb is too low it might skirt the curb". In this paper we assume that the user's concept vocabulary is provided as input, as that problem is orthogonal to the focus of this paper and can be addressed independently.

The most closely related orthogonal research has looked at the problems of learning high-level symbolic models of AI systems using either data from observations or using interventions. For instance, Konidaris, Kaelbling, and Lozano-Perez (2018) assume access to predefined options and learn the high-level symbols that describe those options at the high-level. However, the learned symbols are not directly interpretable and the authors assign meanings to them based on which states and actions they appear in and in what form. Verma, Marpally, and Srivastava (2021) learn user-interpretable models of the AI system using query-answering, but they make the strong assumption that the user's vocabulary is precise enough to distinguish between any two states in a high fidelity simulator. Furthermore, their work assumes that the user and the AI system reason at the same level of abstraction. These assumptions make prior approaches difficult to use in settings where the AI system may use a more detailed model of the environment (e.g., a physics simulator) than the conceptual scope of the user. Another major difference of this paper with prior work is that unlike prior work, the AI system "actions" that our approach learns are not predefined, and in general, tend to be at a much higher level of abstraction than the ones that the AI system uses. E.g., our system may learn and inform the user about actions that can be described at the level of their conceptual vocabulary such as killing a monster in an ATARI game or taking a right turn while the target AI system's true actions may be keystrokes or motor control commands. In this way, the action representation that is learned is specific to the given user. A greater discussion of the related work can be found in Sec. 5.

As a starting point, we begin by assuming determinism and full observability on part of the AI system. We show the working of this approach using the agents based on the General Video Game Artificial Intelligence (GVGAI) framework (Perez-Liebana et al. 2016, 2019).

The rest of this paper is organized as follows. The next section presents some background terminology used in this paper. Section 3 introduces symbol description learning problem and explains our approach to solve it; Section 4 explains the empirical evaluation setup and analyzes the results; Section 5 discusses the relevant literature; and Section 6 highlights our main conclusions.

## 2 Preliminaries

We assume the AI system ("agent" henceforth) has a deterministic black-box policy corresponding to a decision process defined as a 5-tuple $\langle S, A, T, R, G\rangle$, where $S$ is the state space, $A$ is the set of actions that the agent can execute, $T : S \times A \rightarrow S$ is the transition function, $R : S \times A \rightarrow \mathbb{R}$ is the set of rewards associated with each valid transition ($\mathbb{R}$ is the set of real numbers), and $G$ is the set of goal states that the agent is trying to reach. The black-box *deterministic policy* $\Pi : S \rightarrow A$ maps each state to the action that the agent should execute in that state to reach one of the goal states $g \in G$.

This policy need not be optimal, but should be stationary for our approach to learn its correct symbolic description. We now see the descriptions that we will use to represent the agent using this black-box policy.

### 2.1 Symbolic Descriptions

We assume that the user wishes to estimate the agent's internal model as a symbolic description similar to the ones learned by Kansky et al. (2017), James, Rosman, and Konidaris (2020), etc. Such descriptions can be easily interpreted with statements such as "under situations where $X$ hold, if the agent executes actions $a_1, \ldots, a_k$ it would result in $Y$", where $X$ and $Y$ can be expressed using the user-provided concepts. We formally define such symbolic descriptions as follows:

**Definition 1.** A *symbolic description* is represented as a pair $M = \langle \tilde{P}, \tilde{A}\rangle$, where $\tilde{P} = \{\tilde{p}_1, \ldots, \tilde{p}_n\}$ is the finite set of predicates; $\tilde{A} = \{\tilde{a}_1, \ldots, \tilde{a}_k\}$ is a finite set of actions (operators). Each action $\tilde{a}_j \in \tilde{A}$ is represented as a tuple $\langle pre(\tilde{a}_j), \mathit{eff}(\tilde{a}_j)\rangle$, where $pre(\tilde{a}_j)$ represents the set of predicate atoms that must be true in a state where $\tilde{a}_j$ can be applied, $\mathit{eff}(\tilde{a}_j)$ is the set of positive or negative predicate atoms that will change to true or false respectively as a result of execution of the action $\tilde{a}_j$.

Here, each predicate could be absent, positive or negative in the precondition and effects of each action, and cannot be positive (or negative) in both preconditions and effect simultaneously. Such descriptions are referred to as STRIPS-like models (Fikes and Nilsson 1971). We call such models interpretable as they help the user understand what the agent will do in a certain situation, and what the limits and capabilities of the agent are. E.g., such a description in an ATARI game Zelda could indicate that the player must kill all monsters to finish the game, or that the player must be adjacent to a monster to kill it. This is different from the previous attempts for defining interpretability (Barceló et al. 2020; Lipton 2018; Poursabzi-Sangdeh et al. 2021) as those definitions focus only on explaining the representation of the structure learned by the agent, and hence do not explain the agent's working in terms that the user can understand.

As mentioned earlier, our approach supports user's vocabulary that is not precise enough to express all the simulator states. Hence, we use abstraction to model the relationship between the user's concepts and the the agent's state space.

### 2.2 Abstraction

We now define the notion of abstraction used in this work. Several approaches have explored the use of abstraction in planning (Bäckström and Jonsson 2013; Giunchiglia and Walsh 1992; Helmert et al. 2007; Konidaris 2019; Sacerdoti 1974; Srivastava, Russell, and Pinto 2016). We refer to $\tilde{S}$ as the set of *high-level* or *abstract* states, and $S$ as the set of *low-level* or *concrete* states. We define abstraction as in (Srivastava, Russell, and Pinto 2016):

**Definition 2.** Let $S$ and $\tilde{S}$ be sets such that $|\tilde{S}| \preceq |S|$. An *abstraction* from $S$ to $\tilde{S}$ is defined by a surjective function $f : S \rightarrow \tilde{S}$. For any $\tilde{s} \in \tilde{S}$, the concretization function $\gamma_f(\tilde{s}) = \{s \in S : f(s) = \tilde{s}\}$ denotes the set of states represented by the abstract state $\tilde{s}$.

Following this notion, we use ˜ whenever we refer to a state, a predicate, or an action pertaining to the abstract state space. The abstractions we use in this work are forall-exists abstraction (Srivastava, Russell, and Pinto 2016) which are formally defined as follows:

**Definition 3.** An abstraction is a *forall-exists abstraction* iff for every $\tilde{s}' \in \tilde{a}(\tilde{s})$, for every $s \in \tilde{s}$, there exists a $s' \in a(s)$ such that $s' \in \tilde{s}'$.

When we learn the symbolic description $\tilde{M}^*$ of the AI system having a policy $\Pi$, the actions in $\tilde{M}^*$ end up being temporal abstractions of the actions that the AI system can execute. In the context of this work, *temporal abstraction* is an abstract state transition, i.e., it is a transition from one high-level state to another such that there are multiple low-level transitions corresponding to it.

## 3 Learning Symbolic Descriptions

We assume that the input AI system is an autonomous agent with access to a simulator $\mathcal{S}$. The simulator $\mathcal{S}$ is defined as a 3-tuple $\langle S, A, T\rangle$, where $S$ is the state space, $A$ is the set of possible actions, and $T$ is the transition function. We assume that the user understands certain concepts,

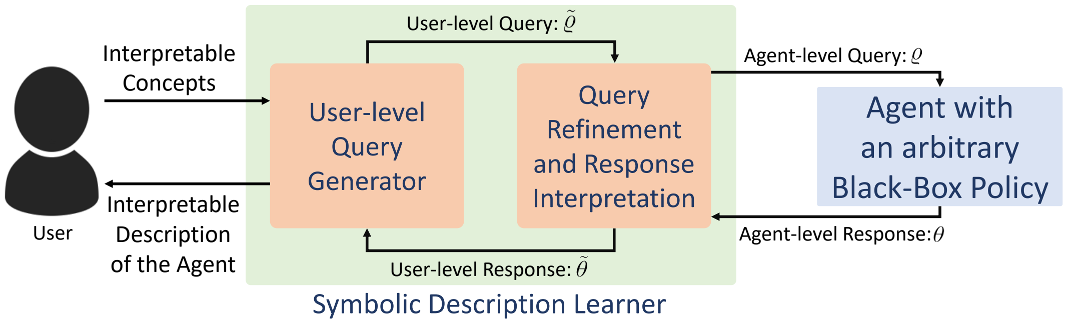

User-level Query: $\tilde{\varrho}$
Interpretable Concepts
User-level Query Generator
Query Refinement and Response Interpretation
Agent-level Query: $\varrho$
Agent with an arbitrary Black-Box Policy
User
Interpretable Description of the Agent
Agent-level Response: $\theta$
User-level Response: $\tilde{\theta}$
Symbolic Description Learner

Figure 1: The symbolic description learner that uses the user's preferred vocabulary, generates the user-level query, converts it to the agent-level query, and based on its responses returns a user-interpretable symbolic description of the agent's capabilities.

which we model as predicates. These concepts may not be sufficient to discern all dynamic processes captured in the simulator. E.g., the user may be unable to distinguish between two distinct simulator states. Hence, there is a discrepancy between what the user and the agent can express. Fig. 1 presents an overview of the approach, the internal components of which are described in this section.

Let the set of predicates that the user understands be $\tilde{P}$, and the set of states expressible using these predicates be $\tilde{S}$. Since the predicates $\tilde{P}$ may not be enough to express all the states in $S$, more than one state in $S$ can correspond to a single state in $\tilde{S}$. This relationship can be modeled as an abstraction where $\tilde{S}$ represents the high-level or abstract states, and $S$ represents the low-level or concrete states. This can be explained using an example from the Zelda domain shown in Fig. 2. Here the simulator state expresses pixel level details of the domain, whereas the state in the user's vocabulary can express only some high-level concepts.

In this work, we assume that the abstraction from $S$ to $\tilde{S}$ is a forall-exists abstraction and all low-level concrete states that map to the same high-level abstract state are strongly connected through low-level actions. This ensures determinism in the transition system at the high-level that results from the abstraction. Now, we formally define what we mean by a symbolic description learning problem:

**Definition 4.** A *symbolic description learning problem* is a 3-tuple $\langle \tilde{P}, \mathcal{A}, \mathcal{S} \rangle$ whose objective is to learn the symbolic description $\tilde{M}^*$ of the agent $\mathcal{A}$, using a simulator $\mathcal{S}$, given the set of user interpretable predicates $\tilde{P}$.

This is similar to the agent interrogation task (Verma and Srivastava 2020; Verma, Marpally, and Srivastava 2021) but without the assumption that the user's concept vocabulary is precise enough to discern all distinct simulator states and that the user is interested in low-level agent actions. These are significant limitations as non-experts would be hard-pressed to have such a precise conceptual vocabulary. E.g., a large number of ATARI game states may be understood as "being in a room with a monster". Furthermore, describing the effect of lowest level of agent actions like keystrokes does not help a user understand the agent's behavior.

The symbolic description $\tilde{M}^* = \langle \tilde{P}, \tilde{A} \rangle$ that we aim to learn consists of user interpretable predicates $\tilde{P}$, and the descriptions of agent's actions in terms of $\tilde{P}$. One major problem that we solve is to generate a list of actions $\tilde{A}$ that can be described using $\tilde{P}$. This is discussed in detail in Sec. 3.1.1.

Another key problem we solve in our approach is to facilitate the interaction between the user and the agent even when they are at different levels of abstractions. We do this by using an active hierarchical query answering approach.

Formally, a *query* is a function that maps an agent to a response. Our algorithm poses the queries to the agent that responds to them using the simulator. The algorithm uses a hierarchical process by first generating the queries in the user's vocabulary – termed as user-level queries– and then refining them to the agent level – termed as agent-level queries. These are formally defined as:

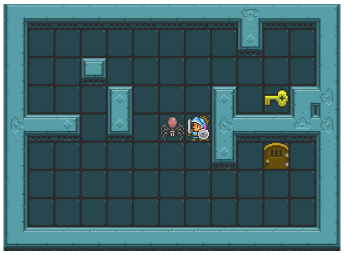
**Definition 5.** A *user-level query* $\tilde{\varrho}$ is a function that maps an agent $\mathcal{A}$ to a response $\tilde{\theta}$, and is parameterized by a state $\tilde{s}_I$, and a plan $\tilde{\pi} = \langle \tilde{a}_1, \dots, \tilde{a}_N \rangle$, where $\tilde{s}_I$ is a high-level state (represented in the user's vocabulary $\tilde{P}$) in which the agent should start to execute the sequence of actions $\tilde{\pi}$, and $\tilde{a}_i \in \tilde{A}$ are the agent actions represented in the user's vocabulary. The response $\tilde{\theta}$ to such query is a tuple $\langle l, \tilde{s}_F \rangle$, where $l$ is the length of the plan $\tilde{\pi}$ that the agent could successfully execute, and $\tilde{s}_F$ is the state the agent was in after executing the last successful action.

**Definition 6.** An *agent-level query* $\varrho$ is a function that maps an agent $\mathcal{A}$ to a response $\theta$, and is parameterized by states $s_I, s_G \in S$, where $S$ is the state space of the agent. The response $\theta$ to such a query is a boolean variable which is $\top$ if the agent can reach state $s_G$ from state $s_I$, and is $\bot$ otherwise.

### 3.1 Learning Symbolic Descriptions

To solve the symbolic description learning problem, we present an approach that iteratively queries an agent and learns the correct symbolic description based on the AI system's responses. A high-level description of the approach is shown in Algorithm 1.

The algorithm takes as input the user interpretable predicates $\tilde{P}$, the agent $\mathcal{A}$, and the simulator $\mathcal{S}$. First, it collects a set of low-level observation traces $O$ from the agent



**Simulator state:**
pixel_1_1(#FD2310)
pixel_1_2(#B24319)
.
.

**Simulator actions (keystrokes):**
W, A, S, D, E

**State in user's vocabulary:**
monster_at(5,3)
player_at(6,3)
key_at(9,4)
door_at(9,2)

**Actions in user's vocabulary (learned):**
kill_monster, go_to_key,
go_to_door, open_door

Figure 2: States and actions represented in possible simulator and user vocabularies for Zelda.

**Algorithm 1** Symbolic Description Learning Algorithm

```
Require: predicates $\tilde{P}$, agent $\mathcal{A}$, simulator $\mathcal{S}$
 1: $O \leftarrow generate\_observations(\mathcal{A}, \mathcal{S})$
 2: $\tilde{A} \leftarrow generate\_HLAs(O)$
 3: Set $\tilde{M}^* = \phi$, $\tilde{L} \leftarrow \{pre, eff\}$
 4: for each $\langle \tilde{L}, \tilde{A}, \tilde{P} \rangle$ do
 5:     Generate $M_+, M_-, M_\emptyset$ by setting $\tilde{P}$ in $\tilde{A}$ at $\tilde{L}$ to
        $+, -$, and $\emptyset$ in $\tilde{M}^*$
 6:     for each pair $M_1, M_2$ in $\{M_+, M_-, M_\emptyset\}$ do
 7:         $\tilde{\varrho} \leftarrow generate\_query(M_1, M_2)$
            //$\tilde{\varrho}$ is of the form $\langle \tilde{s}_0, \tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_k \rangle$
 8:         $s_0 \leftarrow refine\_state(\tilde{s}_0)$
 9:         for $i$ in range $[1, k]$ do
10:             Set $\tilde{s}_i$ by applying $\tilde{a}_i$ in state $\tilde{s}_{i-1}$
11:             $s_i \leftarrow$ concretize state $\tilde{s}_i$          Query Refinement
12:         end for
13:         for $i$ in range $[0, k-1]$ do
14:             $\varrho \leftarrow \langle s_i, s_{i+1} \rangle$
15:             $\theta \leftarrow ask\_agent(\varrho, \mathcal{A}, \mathcal{S})$
16:             if $\theta = \bot$ then
17:                 $\tilde{\theta} \leftarrow \langle i, \tilde{s}_i \rangle$          Response Interpretation
18:             end if
19:         end for
20:         $\tilde{\theta} \leftarrow \langle k, \tilde{s}_k \rangle$
21:         $\tilde{M}^* \leftarrow consistent\_model(\tilde{\theta}, M_1, M_2)$
22:     end for
23: end for
24: return $\tilde{M}^*$
```

(line 1), generated by the agent in the simulator performing some randomly generated task using its policy. These observations have grounded transitions and appear in the form $\langle s_0, s_1, \ldots, s_n \rangle$, and are used to generate the partial action descriptions $\tilde{A}$ in user vocabulary (line 2). This process is described in Sec. 3.1.1. The algorithm then initializes the variable for storing the symbolic description $\tilde{M}^*$ to an empty model, and the variable $\tilde{L}$ to keep track of the locations where a predicate can be added to an action – precondition and effect (line 3).

Then the algorithm iteratively determines how each high-level predicate $\tilde{P}$ appears in an action $\tilde{A}$, in the precondition and the effect (line 4). To achieve this, it generates three models $M_+, M_-$, and $M_\emptyset$ by setting $\tilde{P}$ in $\tilde{A}$ at $\tilde{L}$ to $+, -$, and $\emptyset$ in $\tilde{M}^*$ (line 5). Setting a predicate in an action as $+$ or $-$ in a precondition (or effect) means additng it as a positive or negative precondition (or effect). Setting it to $\emptyset$ means it does not appear in precondition (or effect). Then the algorithm picks two of these models at a time (line 6) and generates an user-level query $\tilde{\varrho}$ that can distinguish between the two models similar to the query generation by Verma, Marpally, and Srivastava (2021) (line 7). These user-level queries cannot be posed directly to the agent, as the user and simulator vocabularies are not the same. Hence, these user-level queries are converted to one or more agent-level queries using the query refinement process (lines 8-14). Then the algorithm uses a response interpretation process to collect the responses of all agent-level queries that correspond to the same user-level query, and process them to generate a single user-level response corresponding to each user-level query (lines 15-20). The query refinement and interpretation processes are described in detail in Sec. 3.1.2.

Finally, the algorithm finds the model consistent with the agent response as in (Verma, Marpally, and Srivastava 2021) (line 21). This includes asking the same query to the two models and checking which model's response conforms to that of the agent. This process is repeated until the algorithm finds how each predicate in $\tilde{P}$ appears in each action in $\tilde{A}$, in the precondition and the effect.

### 3.1.1 Generating a High-Level Action Vocabulary

This component corresponds to line 2 of the algorithm. As mentioned earlier, the set of actions in the user vocabulary are not part of the input, hence the algorithm learns the set of possible high-level actions $\tilde{A}$ based on a set of input observations. To generate the set of high-level actions, we use temporal abstraction. Hence, we define high-level actions as the temporal abstractions of low-level actions that change the high-level state. We call these actions high-level actions (HLAs), and they are generated using the *generate_HLAs(O)* procedure. The *generate_HLAs(O)* procedure converts an input set of low-level observations $O$ generated by abstracting the states (forall-exists abstractions), and noting any abstract state change as a unique high-level action.

We also store the states before and after executing these actions as possible set of precondition and effects. We then create sets of actions that cause similar state transition. Then similar to Stern and Juba (2017), for each of these sets, we create a possible set of preconditions by taking intersection of the predicates that were true in the states where these actions were executed. Similarly, we create possible effects using states after the actions were executed. This gives us partial action descriptions of these high-level actions.

**Lemma 1.** The set of high-level actions $\tilde{A}$ learned using *generate_HLAs(O)* procedure are temporal abstractions of low-level agent actions $A$.

*Proof (Sketch).* High level actions $\tilde{A}$ are learned by performing forall-exists abstraction on the concrete low level states. Whenever the abstract state changes in a transition, it is considered as a high-level action. Assume the transition at high-level corresponding to the action $\tilde{a}_i$ be $\tilde{s}_1 \rightarrow \tilde{s}_2$. Now, according to the abstraction framework defined in Sec. 2.2, all states that are concretization of the same state are strongly connected through low-level actions. Hence, the action $\tilde{a}_i$ is equivalent to temporal abstraction of multiple actions causing transitions within concretizations of $\tilde{s}_1$ followed by a low-level action causing transition between any one of the concretization of $\tilde{s}_1$ and any one of the concretization of $\tilde{s}_2$. □

### 3.1.2 Query Refinement and Response Interpretation Process

Query refinement corresponds to lines 8-14 of the response generation algorithm. It converts a user-level query $\tilde{\varrho}$ to a set of agent-level queries that are posed to the agent. A query $\tilde{\varrho}$ can be represented as a trace $\langle \tilde{s}_0, \tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_N \rangle$, where the initial state $\tilde{s}_I = \tilde{s}_0$ and the plan $\tilde{\pi} = \langle \tilde{a}_1, \tilde{a}_2, \ldots, \tilde{a}_N \rangle$. The first step is to convert the trace to the form $\langle \tilde{s}_0, \tilde{a}_1, \tilde{s}_1, \tilde{a}_2, \tilde{s}_2 \ldots, \tilde{a}_k, \tilde{s}_N \rangle$ using the partial action descriptions $\tilde{A}$ learned earlier from the observations (line 2). Each of the high-level states are then concretized to the low-level states. The resulting consecutive low-level states are paired in the form $\langle s_i, s_{i+1} \rangle$, where $i$ ranges from 0 to $N-1$. These pairs can be directly used as agent-level queries posed to the agent sequentially.

Response Interpretation corresponds to lines 15-20 of the response generation algorithm. The pairs of states $\langle s_i, s_{i+1} \rangle$ are given sequentially to the agent and the agent responds if it can reach from state $s_i$ to $s_{i+1}$ using its internal policy. If the agent responds true for all such pairs, then it shows that the agent can execute the high-level plan successfully, and the final high-level state along with the plan length is set as a response $\tilde{\theta}$ (line 20). However, if the agent fails to reach a state $s_{i+1}$ from the state $s_i$, then it is treated as a failure to execute action $\tilde{a}_i$ in state $\tilde{s}_i$, and the response $\tilde{\theta}$ is set as $\langle i, \tilde{s}_i \rangle$ (line 17). This also results in updating the partial description of the failing high-level action in line 21.

The theoretical results shown below formalize the notions about the properties of learned descriptions.

**Lemma 2.** The response $\tilde{\theta}$ to the user-level query generated by Algorithm 1 is correct given the correct agent responses to all agent-level queries corresponding to the same user-level query.

*Proof (Sketch).* The query refinement process generates the intermediate states when executing a plan, and refines each one of them to one of their possible concretizations. For each pair $\langle s_i, s_{i+1} \rangle$ of consecutive low-level states the response interpretation process asks the agent if it can reach state $s_{i+1}$ from the state $s_i$. If the agent responds affirmatively, then there exists a plan at low-level whose temporal abstraction is equivalent to the high-level action that query refinement process used to generate $\tilde{s}_i$ and $\tilde{s}_{i+1}$. If the agent cannot reach $s_{i+1}$ from $s_i$, then the response generation process reports a failure. In both the cases, the user-level query's response is correct. □

**Theorem 1.** Given a set of observations $O$, and the set of predicates $\tilde{P}$ in the user vocabulary, every observed transition in $O$ is a grounding of at least one action from the set of HLAs $\tilde{A}$.

*Proof (Sketch).* For every transition $\langle s_i, s_{i+1} \rangle$ in $O$, on abstraction to high-level vocabulary $s_i$ and $s_{i+1}$ will map to either same high-level state or different. If they map to different high-level state then there will be a high-level action capturing the transition, because of the way we construct the set of high-level instructions. If these states map to same high-level state, then this transition will be captured as the temporal abstraction of the subsequent action in the trace that maps to a different high-level state (along with all low-level actions in between). Hence, because of the way we construct high-level actions, each of the observation in $O$ maps to at least one high-level action. □

The learned description is sound if all observed transitions in $O$ are grounding of at least one of the learned action in $\tilde{A}$, and the precondition and effect of the action that they correspond to models the state before and after their execution correctly.

**Theorem 2.** The learned description of the capabilities of an agent with deterministic black-box policy is sound given the the predicates $\tilde{P}$ in the user vocabulary and the set of low-level observations $O$ generated in stationary fully observable settings.

*Proof (Sketch).* As shown in Theorem 1, each of the low level observation is captured by at least one of the high-level action. Now the partial preconditions of a high-level action $\tilde{a}_i$ are created by taking intersection of all the states where $\tilde{a}_i$ was successfully executed. Hence the partial preconditions are correct given the observations $O$ as they are valid for each one of them. Same argument is valid for the partial effects as they are constructed similarly to the preconditions. The set of partial preconditions and effects is changed only based on responses of the agent to the user-level queries using the methodology described in Verma, Marpally, and Srivastava (2021), which was shown to always learn the correct descriptions.[1] Hence, the learned descriptions of the agent actions is sound given the user vocabulary $\tilde{P}$ and low-level observations $O$. □

## 4 Empirical Evaluation

We implemented and evaluated Alg. 1 with two types of agents to investigate its applicability on agents using

[1] Theorem 3 in Verma, Marpally, and Srivastava (2021)

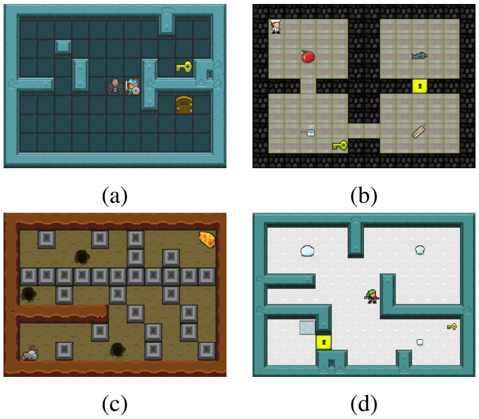


Figure 3: GVGAI's domains; (a) Zelda, (b) Cook-Me-Pasta, (c) Escape, and (d) Snowman.

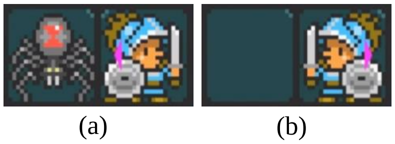
arbitrary internal planning and reasoning paradigms. *Policy agents* use (possibly learned) black-box policies to respond to agent-level queries (Def. 6). We used policy agents with hand-coded policies for evaluating Alg. 1. *Search agents* respond to agent-level queries using search algorithms. Our implementation of search agents for evaluation uses A$^*$ search. We now describe the setup used for the experiments.

### 4.1 Experimental Setup

In the context of this work, the agents are based on the General Video Game Artificial Intelligence (GVGAI) framework (Perez-Liebana et al. 2016, 2019). We modified the games to make them stationary and deterministic. E.g., in Zelda domain, we set the monsters as stationary. We performed experiments on four domains as shown in Fig. 3; Zelda, Cook-Me-Pasta, Escape, and Snowman. Each of these domains has a grid like structure, and the agent has to perform a set of actions to finish the game.

**Zelda.** The Zelda domain, as shown in Fig. 3a, consists of a key, the player, and monsters. To finish the game, all the monsters must be killed and the key must be used to open a door to escape. The player can move one cell at a time in the direction it is facing. If the player moves into the cell containing the key, the player picks up the key by executing the low-level action `E` (special keystroke). The same key is used to kill the monster when the player is facing the monster and is in a cell adjacent to the monster, and to escape when the player is in a cell adjacent to the door and facing it.

**Cook-Me-Pasta.** The Cook-Me-Pasta domain, as shown in Fig. 3b, consists of raw pasta, sauce, boiling water, tuna (fish), lock, and key. The objective is to cook tuna pasta using a three step process. First the pasta is cooked by adding boiling water to the raw pasta, this can be done by pressing `E` while holding both the ingredients. Similarly tuna is cooked by mixing sauce and tuna. Finally, the cooked pasta and the cooked tuna are to be mixed together. One or more of the ingredients can be locked in a room which must be opened using a key.



```
Keystrokes 1: W→A→E
Keystrokes 2: S→S→A→W→W→A→E
```

(c)

```
(:action a9
 :parameters ()
 :precondition (and (at p1 6-3)
                    (at m1 5-3)
                    (monster_alive m1)
                    (next_to_monster))
 :effect (and (clear 5-3)
              (not (at m1 5-3))
              (not (monster_alive m1))
              (not (next_to_monster))))
```

(d)

```
If the player p1 is at cell (6-3), the
monster m1 is alive, at cell (5-3) and
next to p1, then the agent can  act to
reach  a  state where m1 is  not alive,
cell (5-3) is empty and p1 is not next
to any monster.
```

(e)

Figure 4: Learning the *kill monster* action of Zelda. Subfigures (a) and (b) show the simulator states immediately before and after executing either of the keystroke sequences shown in (c). Alg. 1 generates and uses such sequences to learn a user-interpretable symbolic description of a high-level action (d) for killing a monster when the agent is in an arbitrary orientation next to it. This action captures keystroke sequences such as those in (c) as possible implementations or refinements. (e) A boilerplate readable description that can be generated from the learned description.

**Escape.** The Escape domain, as shown in Fig. 3c, consists of movable blocks, fixed holes, and cheese. The blocks can be pushed into the holes to clear out a path. The game is finished when the player reaches the location with cheese.

**Snowman.** The Snowman domain, as shown in Fig. 3d, consists of three pieces of a snowman: the top, middle, and bottom piece; a key that can be used to unlock a door (like other domains), and the goal cell. The objective of the game is to assemble the snowman in the goal location in order, constrained by the player being able to hold only one piece at any given time.

**User-Interpretable Vocabulary.** The user interpretable vocabularies for each of the four domains are shown in Table 1. The semantics of each of these predicates is available in the supplementary material. Note that information like orientation of the agent (player) in each of these domains is not captured by any of the predicates. This information

Table 1: Predicates included in the user vocabulary for each test domain.

| Zelda | Cook-Me-Pasta | Escape | Snowman |
|---|---|---|---|
| at(?ob ?loc) | at(?ob ?loc) | at(?ob ?loc) | at(?ob ?loc) |
| wall(?loc) | wall(?loc) | wall(?loc) | wall(?loc) |
| clear(?loc) | clear(?loc) | clear(?loc) | clear(?loc) |
| has_key() | has_key() | is_hole(?loc) | has_key() |
| escaped() | pasta_cooked() | is_goal(?loc) | player_has(?ob) |
| monster_alive(?m) | is_door(?loc) | is_block(?loc) | is_goal(?loc) |
| next_to_monster() | | escaped() | placed(?part) |
| | | | is_door(?loc) |

is important for the low-level policies as certain actions can only be executed in certain orientations. E.g., in Zelda domain, the player must face the monster when killing it.

The complete list of capabilities of an agent may be irrelevant to a user's current needs. Without loss of generality, our implementation allows the input to include sets of formulas representing the properties that may be of interest to the user. This set can be the set of all grounded predicates in the user's conceptual vocabulary. We then query the agent in a way that learns descriptions of actions (agent capabilities) that are relevant for achieving this set of input properties.

For each domain, and for each grid size in that domain, we create a random game instance with the goal as achieving one of the user's specified properties of interest. The number of walls in all four domains is set to 20% of the total cells in the grid, whereas all other objects are generated randomly. We use the solution to that instance to generate the low-level trace that is used in lines 1-2 of the algorithm. These solutions are not always optimal. All the experiments are run on 5 GHz Intel i9 CPUs with 64 GB RAM and Ubuntu 18.04.

### 4.2 Results

The symbolic description learner learns the correct description of all high-level actions $\tilde{A}$ in a domain. Fig.4 shows one such learned description from the Zelda domain. Full learned models for all four domains are included in the supplementary material. Note that the cell locations mentioned correspond to the game state in Fig. 2a. The learned action names are random (a9 in Fig.4c), and this can be interpreted as the kill monster action. Fig.4d shows a boilerplate readable description that can be generated from the learned description. Note that this readable description generation is not part of this work.

To analyze the effect of increasing the size of the state space on the performance of the search and policy agents, we vary the grid size and analyze its affect. Fig. 5 shows the graphs for the experimental runs on the four domains. In all four domains, for both kinds of agents, the number of queries increases as we increase the grid size. This happens because the queries are grounded, and the number of possible groundings of the predicates increases as we increase the grid size. The number of queries required by the policy agent is higher than that of the search agent in most cases. This is because large number of user-level queries fail to run successfully on the agent as the plan in the user-level query does not always align with the policy of the agent. However, the time per query is lesser for the policy agents as they can answer the queries by following their policy, whereas the search agents perform an exhaustive search of the state space for every agent-level query.

**Correctness.** Since there is no ground-truth model to check the accuracy of the learned model, we empirically check if the learned model conforms to all observed traces. This means for each of the transition in the observed traces, we check if the transition is a grounding of at least one of the learned actions. For all four domains, we did not find any instance where this was not the case.

## 5 Related Work

Konidaris, Kaelbling, and Lozano-Perez (2018) present a paradigm for learning high-level propositional models of options representing various "skills." James, Rosman, and Konidaris (2020) extend this approach to learn models in a form that is generalizable to multiple tasks. These existing approaches are complementary to the presented work. While they use options or skills as inputs to learn models defining when those skills will be useful in terms of auto-generated symbols (for which explanatory semantics could be derived in a post-hoc fashion), our approach uses user-provided interpretable concepts as a priori inputs to learn agent capabilities: high-level actions as well as their relational, interpretable descriptions in terms of the input vocabulary.

Zhang et al. (2018) learn symbolic transition models and use them for planning using a set of input attributes. These attributes are interpretable and the set of actions are learned in the form of transitions using random walks in the environment. This needs extensive hand-coding as each of the state must manually be assigned all attributes that are present or true in that state. Additionally, the set of attributes is given as input which does not take into account user preferences.

Kansky et al. (2017) learn symbolic models of simulators based on ATARI games by learning action effects by using conjunctions of binary input features. Some other approaches learn models using symbolic physics engine parameters and graph neural networks (Battaglia et al. 2016; Cranmer et al. 2020). Through their description language, they can generalize over multiple tasks using the learned entities and operators. Agrawal et al. (2016) and Fragkiadaki et al. (2016) use convolutional neural networks to learn intuitive physical models of object interaction. These approaches need a lot of training data to learn the correct

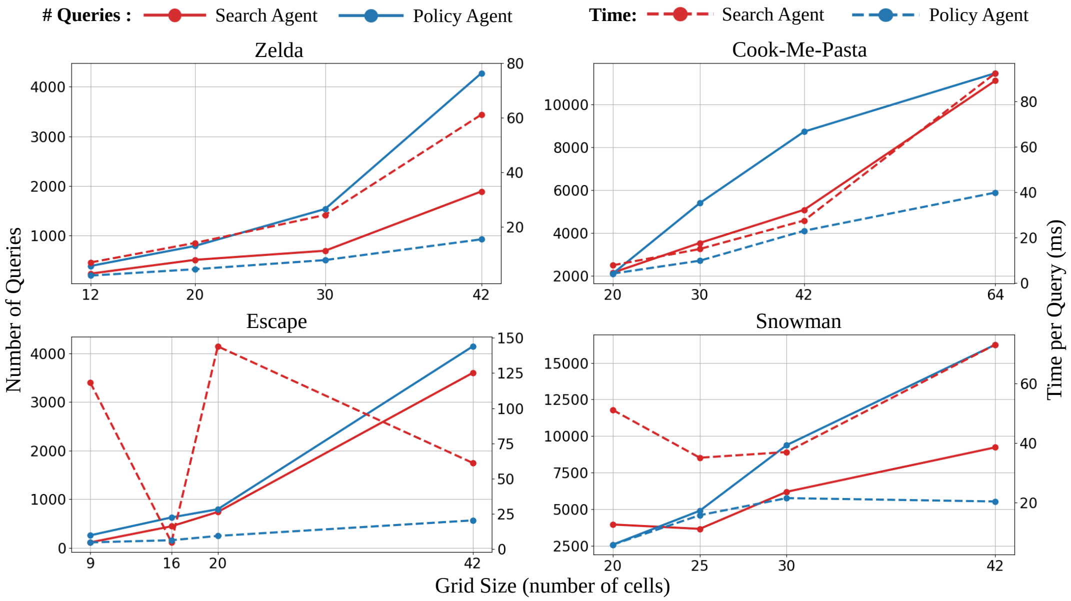


Figure 5: Performance comparison of search-based agents and policy-based agents in terms of the number of queries asked and time taken per query when increasing the grid size.

model in their chosen representations. The same problem is faced by approaches that create interpretable descriptions of reinforcement learning policies using trees (Liu et al. 2018) or specialized programming languages (Verma et al. 2018). Unlike these approaches our method needs minimal data and can generalize using relational transitions captured by the symbolic models that we learn.

The planning community has also worked on learning symbolic models of agents similar to the ones that we learn in this work. Jiménez et al. (2012) and Arora et al. (2018) present comprehensive review of such approaches. These methods make broad assumptions that the agent model is internally expressed in the same vocabulary as the user's (Gil 1994; Weber, Morwood, and Bryce 2011), or at a similar level of abstraction (Mehta, Tadepalli, and Fern 2011; Verma, Marpally, and Srivastava 2021; Nayyar, Verma, and Srivastava 2021). Our approach is able to learn the symbolic descriptions in terms of fewer concepts than used by the agent.

## 6 Conclusion

We presented a novel approach for learning the symbolic description of an AI system in terms of user-interpretable concepts through active query answering. Our approach works for settings where the user's conceptual vocabulary is imprecise and cannot directly express the agent's capabilities. Our empirical analysis showed that we can successfully learn these descriptions for agents internally using black-box deterministic policies, and for planning agents using search techniques. Extending this approach for partially observable settings, relaxing the various assumptions we made, and performing a more comprehensive complexity analysis like Verma and Srivastava (2021) are some of the promising future directions for this work.

## Broader Impact

In the recent past, learning interpretable descriptions of the capabilities of an AI system is one of the main focus areas of AI research. Our work would enable people with different level of knowledge and concepts to understand the capabilities of an AI system in terms that they can interpret and to assess if such systems are safe to work with. Our approach provides correctness guarantees of learning user-interpretable symbolic descriptions given the access to finite observation traces of the agent executing its black-box policy in a simulator. If the simulator is susceptible to errors, our approach might correspondingly generate incorrect descriptions. This issue can be mitigated using formal verification of the simulator.

## Acknowledgements

This work was supported in part by the NSF grants IIS 1844325, OIA 1936997, and the ONR grant N00014-21-1-2045.

## References

Agrawal, P.; Nair, A. V.; Abbeel, P.; Malik, J.; and Levine, S. 2016. Learning to Poke by Poking: Experiential Learning of Intuitive Physics. In *Proc. NIPS*.

Anjomshoae, S.; Najjar, A.; Calvaresi, D.; and Främling, K. 2019. Explainable Agents and Robots: Results from a Systematic Literature Review. In *Proc. AAMAS*.

Arora, A.; Fiorino, H.; Pellier, D.; Métivier, M.; and Pesty, S. 2018. A Review of Learning Planning Action Models. *The Knowledge Engineering Review* 33: E20.

Bäckström, C.; and Jonsson, P. 2013. Bridging the Gap Between Refinement and Heuristics in Abstraction. In *Proc. IJCAI*.

Barceló, P.; Monet, M.; Pérez, J.; and Subercaseaux, B. 2020. Model Interpretability through the Lens of Computational Complexity. In *Proc. NeurIPS*.

Barredo Arrieta, A.; Díaz-Rodríguez, N.; Del Ser, J.; Bennetot, A.; Tabik, S.; Barbado, A.; Garcia, S.; Gil-Lopez, S.; Molina, D.; Benjamins, R.; Chatila, R.; and Herrera, F. 2020. Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI. *Information Fusion* 58: 82–115.

Battaglia, P.; Pascanu, R.; Lai, M.; Jimenez Rezende, D.; and Kavukcuoglu, K. 2016. Interaction Networks for Learning about Objects, Relations and Physics. In *Proc. NIPS*.

Chakraborti, T.; Sreedharan, S.; Zhang, Y.; and Kambhampati, S. 2017. Plan Explanations as Model Reconciliation: Moving Beyond Explanation as Soliloquy. In *Proc. IJCAI*.

Cranmer, M.; Sanchez Gonzalez, A.; Battaglia, P.; Xu, R.; Cranmer, K.; Spergel, D.; and Ho, S. 2020. Discovering Symbolic Models from Deep Learning with Inductive Biases. In *Proc. NeurIPS*.

Dhurandhar, A.; Chen, P.-Y.; Luss, R.; Tu, C.-C.; Ting, P.; Shanmugam, K.; and Das, P. 2018. Explanations based on the Missing: Towards Contrastive Explanations with Pertinent Negatives. In *Proc. NeurIPS*.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2(3-4): 189–208.

Fragkiadaki, K.; Agrawal, P.; Levine, S.; and Malik, J. 2016. Learning Visual Predictive Models of Physics for Playing Billiards. In *Proc. ICLR*.

Gil, Y. 1994. Learning by Experimentation: Incremental Refinement of Incomplete Planning Domains. In *Proc. ICML*.

Giunchiglia, F.; and Walsh, T. 1992. A Theory of Abstraction. *Artificial Intelligence* 57(2-3): 323–389.

Helmert, M.; Haslum, P.; Hoffmann, J.; et al. 2007. Flexible Abstraction Heuristics for Optimal Sequential Planning. In *Proc. ICAPS*.

James, S.; Rosman, B.; and Konidaris, G. 2020. Learning Portable Representations for High-Level Planning. In *Proc. ICML*.

Jiménez, S.; De La Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A Review of Machine Learning for Automated Planning. *The Knowledge Engineering Review* 27(4): 433–467.

Kansky, K.; Silver, T.; Mély, D. A.; Eldawy, M.; Lázaro-Gredilla, M.; Lou, X.; Dorfman, N.; Sidor, S.; Phoenix, S.; and George, D. 2017. Schema Networks: Zero-shot Transfer with a Generative Causal Model of Intuitive Physics. In *Proc. ICML*.

Konidaris, G. 2019. On the Necessity of Abstraction. *Current Opinion in Behavioral Sciences* 29: 1–7.

Konidaris, G.; Kaelbling, L. P.; and Lozano-Perez, T. 2018. From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning. *Journal of Artificial Intelligence Research* 61(1): 215–289.

Lipton, Z. C. 2018. The Mythos of Model Interpretability: In Machine Learning, the Concept of Interpretability is Both Important and Slippery. *ACM Queue* 16(3): 31–57.

Liu, G.; Schulte, O.; Zhu, W.; and Li, Q. 2018. Toward Interpretable Deep Reinforcement Learning with Linear Model U-Trees. In *Proc. ECML PKDD*.

Mehta, N.; Tadepalli, P.; and Fern, A. 2011. Autonomous Learning of Action Models for Planning. In *Proc. NIPS*.

Nayyar, R. K.; Verma, P.; and Srivastava, S. 2021. Differential Assessment of Black-Box AI Agents.

Perez-Liebana, D.; Liu, J.; Khalifa, A.; Gaina, R. D.; Togelius, J.; and Lucas, S. M. 2019. General Video Game AI: A Multitrack Framework for Evaluating Agents, Games, and Content Generation Algorithms. *IEEE Transactions on Games* 11(3): 195–214.

Perez-Liebana, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; and Lucas, S. 2016. General Video Game AI: Competition, Challenges and Opportunities. In *Proc. AAAI*.

Poursabzi-Sangdeh, F.; Goldstein, D. G.; Hofman, J. M.; Wortman Vaughan, J. W.; and Wallach, H. 2021. Manipulating and Measuring Model Interpretability. In *Proc. CHI*.

Randazzo, R. 2018. What went wrong with Uber's Volvo in fatal crash? Experts shocked by technology failure. *The Arizona Republic* .

Sacerdoti, E. D. 1974. Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence* 5(2): 115–135.

Srivastava, S.; Russell, S.; and Pinto, A. 2016. Metaphysics of Planning Domain Descriptions. In *Proc. AAAI*.

Stern, R.; and Juba, B. 2017. Efficient, Safe, and Probably Approximately Complete Learning of Action Models. In *Proc. IJCAI*.

Verma, A.; Murali, V.; Singh, R.; Kohli, P.; and Chaudhuri, S. 2018. Programmatically Interpretable Reinforcement Learning. In *Proc. ICML*.

Verma, P.; Marpally, S. R.; and Srivastava, S. 2021. Asking the Right Questions: Learning Interpretable Action Models Through Query Answering. In *Proc. AAAI*.

Verma, P.; and Srivastava, S. 2020. Learning Generalized Models by Interrogating Black-Box Autonomous Agents. In *AAAI 2020 Workshop on Generalization in Planning*.

Verma, P.; and Srivastava, S. 2021. Learning Causal Models of Autonomous Agents using Interventions. In *IJCAI 2021 Workshop on Generalization in Planning*.

Weber, C.; Morwood, D.; and Bryce, D. 2011. Goal-Directed Knowledge Acquisition. In *ICML 2011 Workshop on Planning and Acting with Uncertain Models*.

Zhang, A.; Sukhbaatar, S.; Lerer, A.; Szlam, A.; and Fergus, R. 2018. Composable Planning with Attributes. In *Proc. ICML*.