Discovering User-Interpretable Capabilities of Black-Box Planning Agents

Pulkit Verma, **Shashank Rao Marpally**, and **Siddharth Srivastava**

Autonomous Agents and Intelligent Robots Lab,
School of Computing and Augmented Intelligence, Arizona State University, USA

{verma.pulkit, smarpall, siddharths}@asu.edu

Abstract

Several approaches have been developed for answering users' specific questions about AI behavior and for assessing their core functionality in terms of primitive executable actions. However, the problem of summarizing an AI agent's broad capabilities for a user is comparatively new. This paper presents an algorithm for discovering from scratch the suite of high-level "capabilities" that an AI system with arbitrary internal planning algorithms/policies can perform. It computes conditions describing the applicability and effects of these capabilities in user-interpretable terms. Starting from a set of user-interpretable state properties, an AI agent, and a simulator that the agent can interact with, our algorithm returns a set of high-level capabilities with their parameterized descriptions. Empirical evaluation on several game-based scenarios shows that this approach efficiently learns descriptions of various types of AI agents in deterministic, fully observable settings. User studies show that such descriptions are easier to understand and reason with than the agent's primitive actions.

1 Introduction

AI systems are rapidly developing to an extent where their users may not understand what they can and cannot do safely. In fact, the limits and capabilities of many AI systems are not always immediately clear even to the experts, as they may use black box policies, e.g., ATARI game-playing agents (Greydanus et al. 2018), text summarization tools (Paulus, Xiong, and Socher 2018), mobile manipulators (Popov et al. 2017), etc.

Ongoing research on the topic focuses on the significant problem of how to answer users' questions about the system's behavior while assuming that the user and AI share a common action vocabulary (Chakraborti et al. 2017; Dhurandhar et al. 2018; Anjomshoae et al. 2019; Barredo Arrieta et al. 2020). Furthermore, most non-experts hesitate to ask questions about new AI tools (Mou and Xu 2017) and often do not know which questions to ask for assessing 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. Lack of understanding about 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 AI systems.

This work presents a new approach for *discovering* capabilities of a black-box AI system. The AI system may use arbitrary internal models, representations, and processes for computing solutions to user-assigned tasks. Prior work on the topic addresses complementary problems of deriving symbolic descriptions for pre-defined skills (Konidaris, Kaelbling, and Lozano-Perez 2018) and of learning users' conceptual vocabularies (Kim et al. 2018; Sreedharan et al. 2022). However, they do not address the problem of *discovering* high-level user-interpretable capabilities that an agent can perform using arbitrary, internal behavior synthesis algorithms (see Sec. 5 for a greater discussion).

As a starting point, in this paper, we assume determinism and full observability on part of the AI system. Since there are no solution approaches for solving the problem even in this foundational setting, our framework can serve as a foundation for solutions to the more general setting in the future.

Running example Consider a game based on "The Legend of Zelda" (Fig. 1) featuring a protagonist player *Link* who must defeat the antagonist monster *Ganon*, and escape through the door using a key. (Fig. 1)(a) shows the game state as the agent sees it; its primitive actions are keystrokes as shown in (b). These keystrokes do not help convey the agent's capabilities because (i) they are too fine-grained, and (ii) they show the set of actions available to the AI system, although its true capabilities depend on its AI planning and learning algorithms. Fig. 1(c) shows common English terms that a user might understand (called the *user's vocabulary*), and the types of capabilities that they may want to know about. Fig. 1(d) shows a parameterized capability discovered by our method. Intuitively, Fig. 1(d) captures the "defeat Ganon" capability.

This paper shows how we can discover and describe an agent's capabilities in the form of Fig. 1(d). This capability description can be readily transcribed as "If the *player1* is at *cell1*; the *monster1* is at *cell2*; the *monster1* is *alive* (not defeated); and the *monster1* is next to the *player1*; then the *player1* can act to reach a state where *cell2* is empty; the *monster1* is *not alive* (defeated); the *monster1* is *not at cell2*; and the *player1* is not next to the *monster1*." Our empirical evaluation shows that our system effectively discovers such high-level capabilities; our user study shows that the discovered capabilities help users effectively estimate black-box
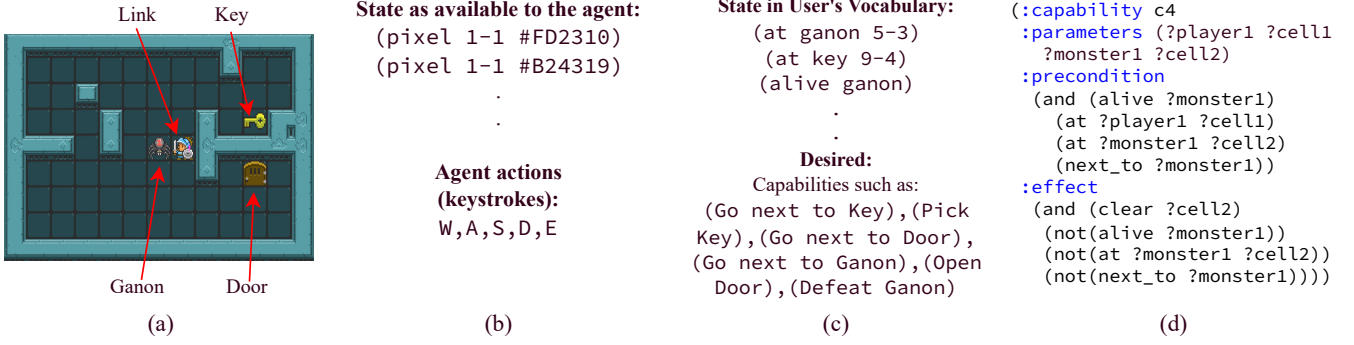
Figure 1: From pixels to interpretable capabilities. (a) A Zelda-like game; (b) States available to the agent and its actions; (c) States represented in user vocabulary, and possible set of desired capabilities; (d) A parameterized capability description learned by our method.

agent capabilities.

The rest of this paper is organized as follows. The next section presents a formal framework for capabilities as well as notions of correctness for discovered agent capabilities. Sec. 3 describes our main algorithms and their formal properties and Sec. 4 presents empirical results and results from user studies. Sec. 5 discusses the relationship of the presented methods with prior work. Finally, Sec. 6 presents our conclusion and future directions.

2 Formal Framework

We model an AI system ("agent" henceforth) as a 3-tuple $\langle S, A, T \rangle$, where S is the state space, A is the set of actions that the agent can execute, $T : S \times A \to S$ is a deterministic black-box transition function determining the effects of the agent's primitive actions on the environment. For brevity of notation, we use $a(s)$ to represent $T(s, a)$, where $a \in A$, and $s \in S$. Given a goal set $G \subseteq S$, a black-box *deterministic policy* $\Pi : S \to A$ maps each state to the action that the agent should execute in that state to reach a $g \in G$.

In this paper, we use "actions" to refer to the core *functionality* of the agent, denoting the agent's decision choices, or primitive actions that the agent could execute (e.g., keystrokes in our running example). In contrast, we use the term "capabilities" to refer to the *high-level behaviors* that the agent can perform using its AI algorithms for behavior synthesis, including planning and learning (e.g., defeating Ganon or picking up the key). Thus, actions refer to the set of choices that a tabular-rasa agent may possess, while capabilities are a result of its agent function (Russell 1997) and can change as a result of algorithmic updates even as the agent uses the same actions.

2.1 Abstraction

We now define the notion of abstraction used in this work. Several approaches have explored the use of abstraction in planning (Sacerdoti 1974; Giunchiglia and Walsh 1992; Helmert et al. 2007; Bäckström and Jonsson 2013; 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 1. Let S and $\tilde{S}$ be sets such that $|\tilde{S}| \leq |S|$. An *abstraction* from S to $\tilde{S}$ is defined by a surjective function $f : S \to \tilde{S}$. For any $\tilde{s} \in \tilde{S}$, the concretization function $f^{-1}(\tilde{s}) = \{s \in S : f(s) = \tilde{s}\}$ denotes the set of states represented by the abstract state $\tilde{s}$.

Following this, we use $\tilde{}$ whenever we refer to a state, a predicate, or an action pertaining to the abstract state space.

2.2 Capability Descriptions

We express capability descriptions using a STRIPS-like representation (Fikes and Nilsson 1971; McDermott et al. 1998). This is because, when used with a user's vocabulary, such a representation can be readily transcribed into statements such as "in situations where X holds, if the agent executes actions $a_1, \ldots, a_k$ it would result in Y", where X and Y are in the user's vocabulary (Camacho and McIlraith 2019; Verma, Marpally, and Srivastava 2021). Such representations have been shown to be intuitive for humans in understanding deliberative behaviors of other agents (Malle 2004; Miller 2019). In our running example, such a description could indicate that if Link is next to Ganon then Link can defeat it. We now formally define a capability.

Definition 2. Given a set of objects $\tilde{O}$; and a finite set of predicates $\tilde{P} = \{\tilde{p}_1^{k_1}, \ldots, \tilde{p}_n^{k_n}\}$ with arities k_i; a *grounded capability* $\tilde{c}^*$ is defined as a tuple $\langle pre(\tilde{c}^*), eff(\tilde{c}^*) \rangle$ where precondition $pre(\tilde{c}^*)$ and effect $eff(\tilde{c}^*)$ are conjunctions of literals over $\tilde{P}$ and $\tilde{O}$.

We also refer to the tuple $\langle \tilde{c}^*, pre(\tilde{c}^*), eff(\tilde{c}^*) \rangle$ as the *capability description* for a capability $\tilde{c}^*$. Here each atom could be absent, positive, or negative (henceforth referred to as the *mode*) in the precondition and the effect of an action. However, we disallow atoms to be positive (or negative) in both the preconditions and the effects of an action simultaneously to avoid redundancy. Semantics of capabilities are close to those of STRIPS actions, but they address vocabulary disparity: an agent can execute a capability $\tilde{c}^*$ in any concrete state s where $\tilde{s} \models pre(\tilde{c}^*)$; as a result, the system reaches a concrete state s' (a member of an abstract state $\tilde{s}'$). Atoms that don't appear in $eff(\tilde{c}^*)$ retain their truth values from $\tilde{s}$ in $\tilde{s}'$ while others are set to their modes (positive,

negative, or absent) in $eff(\tilde{c}^*)$, i.e., $\forall \ell \in eff(\tilde{c}^*)$, $\tilde{s}' \models \ell$. For brevity, we represent this as $\tilde{s}' = \tilde{c}^*(\tilde{s})$.

We refer to the capabilities defined in Def. 2 as grounded capabilities as they are instantiated with a specific set of objects in $\tilde{O}$. We use * whenever we refer to a grounded capability. We define a lifted form of capabilities as parameterized capabilities.

Definition 3. Given a set of objects $\tilde{O}$; a finite set of predicates $\tilde{P} = \{\tilde{p}_1^{k_1}, \ldots, \tilde{p}_n^{k_n}\}$ with arities k_i; a *parameterized capability* $\tilde{c}$ is defined a 3-tuple $\langle args(\tilde{c}), pre(\tilde{c}), eff(\tilde{c}) \rangle$ where $args(\tilde{c})$ is the set of arguments that can be initialized with a set of objects $\tilde{o} \subseteq \tilde{O}$; and $pre(\tilde{c})$ and $eff(\tilde{c})$ are sets of literals over $\tilde{P}$ and $args(\tilde{c})$.

A set of parameterized capabilities constitutes a parameterized capability model. Formally, a *parameterized capability model* is a tuple $\tilde{M} = \langle \tilde{P}, \tilde{C} \rangle$, where $\tilde{P}$ is a finite set of predicates, and $\tilde{C}$ is a finite set of parameterized capabilities.

Our objective is to develop a capability discovery algorithm that learns a parameterized capability model of a black-box AI agent using as input (i) the agent, (ii) a compatible simulator using which the agent can simulate its primitive action sequences, and (iii) the user's concept vocabulary, which may be insufficient to express the simulator's state representation. Such assumptions on the agent are common. In fact, the use of third-party simulators for development and testing is the bedrock of most of the research on taskable AI systems today (including game-playing AI, autonomous cars, and factory robots). Providing simulator access for assessment is reasonable as it would allow AI developers to retain freedom and proprietary controls on internal software while supporting calls for assessment and regulation using approaches such as ours.

Our user studies show the efficacy of this approach using spoken English terms for concepts without an explicit process for vocabulary synchronization. Several threads of ongoing research address the problem of identifying user-specific concept vocabularies (e.g., Kim et al. (2018), Sreedharan et al. (2022)), and the field of intelligent tutoring systems develops methods for helping users understand a fixed concept vocabulary. These methods can be used to either elicit or impart a vocabulary for a given user and such systems can be used to complement the methods developed in this paper.

However, since the problem of capability discovery is not well understood even in settings where user-concept definitions are readily available, we focus on capability discovery with a given vocabulary with known definitions and formalize our approach using them. Furthermore, our empirical evaluation and user studies don't place requirements on user concept vocabularies and show the efficacy of this representation. We formalize these concept definitions as follows:

Definition 4. Given a concrete state $s \in S$, a set of objects $\tilde{O}$ and their tuples $\tilde{O}^{\leq d}$ (of dimension at most d, where d is a positive integer), a set of *concepts/predicates* $\tilde{P} = \tilde{p}_1^{k_1}, \ldots, \tilde{p}_n^{k_n}$ with their arities k_i and an associated Boolean evaluation function $\psi_{\tilde{p}_i} : S \times \tilde{P} \times \tilde{O}^{\leq max(k_i)} \rightarrow \{T, F\}$, $j \leq max(k_i)$, we define $s \models_{\psi_{\tilde{p}_i}} \tilde{p}_i(\tilde{o}_1, \ldots, \tilde{o}_j)$

Algorithm 1: Interactive Capability Model Learning

Input : predicates $\tilde{P}$, agent $\mathcal{A}$
Output : $\tilde{M}$

1   $E \leftarrow$ generate_execution_traces($\mathcal{A}$)
2   $\tilde{C}^* \leftarrow$ generate_partial_capability_descriptions(E)
3   $\tilde{C}' \leftarrow$ parameterize_partial_capabilities($\tilde{C}^*$)
4   $\tilde{M} \leftarrow$ generate_parameterized_capability_model($\tilde{C}'$)
5   Set $\tilde{L} \leftarrow \{pre, eff\}$
6   **for** each $\langle \tilde{L}, \tilde{C}, \tilde{P} \rangle$ in $\tilde{M}$ **do**
7   Generate $\tilde{M}_+, \tilde{M}_-, \tilde{M}_\emptyset$ by setting $\tilde{P}$ in $\tilde{C}$ at $\tilde{L}$ to $+, -, \emptyset$ in $\tilde{M}$
8   **for** *each pair* $\tilde{M}_x, \tilde{M}_y$ *in* $\{\tilde{M}_+, \tilde{M}_-, \tilde{M}_\emptyset\}$ **do**
9         $\tilde{q} \leftarrow$ generate_query($\tilde{M}_1, \tilde{M}_2$)
10        $\tilde{\varrho} \leftarrow$ generate_waypoints($\tilde{q}$)
11       $\varrho \leftarrow$ refine_waypoints($\tilde{\varrho}, \tilde{P}$)
12   **for** i in range$[0, k-1]$ **do**
13          $\theta \leftarrow$ ask_agent($\mathcal{A}, \langle s_i, s_{i+1} \rangle$)
14   break if $\theta = \bot$
15       $\tilde{M} \leftarrow$ consistent_description($i, \tilde{s}_i, \tilde{M}_x, \tilde{M}_y$)

16 **return** $\tilde{M}$

---

iff $\psi_{\tilde{p}_i}(s, \tilde{p}_i, \tilde{o}_1, \ldots, \tilde{o}_j) = T$. We define *the abstraction* $\tilde{s}_{\tilde{P}, \tilde{O}}$ *of a state* $s \in S$ as the set of all literals over $\tilde{P}$ and $\tilde{O}$ that are true in s. $\tilde{S}_{\tilde{P}, \tilde{O}}$ denotes the abstract state space $\{\tilde{s}_{\tilde{P}, \tilde{O}} : s \in S\}$.

We omit subscripts $\tilde{P}$ and $\tilde{O}$ unless needed for clarity. As mentioned, we assume availability of an evaluation function $\psi_{\tilde{p}}$ associated with each predicate $\tilde{p} \in \tilde{P}$. E.g., for a 3-D Blocksworld simulator with objects a and b, and coordinates x, y, and z, "$on(a, b)$ is true exactly for states where $z(a) > z(b)$, $x(a) = x(b)$, and $y(a) = y(b)$." As this example illustrates, such vocabularies can be inaccurate. The abstraction function f (Def. 4) can be modeled as a conjunction of these evaluation functions $\psi_{\tilde{p}}$. We now discuss how we discover capabilities and learn their descriptions.

3 Active Capability Discovery

Our overall approach consists of two main phases: (1) discovering candidate capabilities and their partial descriptions from a set of execution traces of the agent's behavior (Sec. 3.1); and (2) completing the descriptions of the candidate capabilities discovered in step (1) by asking the agent queries designed to assess under which conditions it can execute those capabilities and what their effects are (Sec. 3.2). The interactive Capability Model Learning (iCaML) algorithm (Alg. 1) performs both these steps using user interpretable predicates $\tilde{P}$ and the agent $\mathcal{A}$ as inputs. We now explain these two phases in detail.

3.1 Discovering Candidate Partial Capabilities

Generating execution traces As a first step, Alg. 1 collects a set of execution traces E from the agent (line 1). An *execution trace* e is a sequence of states of the form $\langle s_0, s_1, \ldots, s_{n-1}, s_n \rangle$, such that $\forall j \in [1, n]$ $\exists a_i \in A$ $a_j(s_{j-1}) = s_j$. To obtain the traces $e \in E$, a set of random tasks of the form $\langle s_I, s_G \rangle$, where $s_I, s_G \in S$, are given to the agent $\mathcal{A}$, and the agent is asked to reach s_G from s_I. Intermediate states that the agent goes through form the set of execution traces E.

Discovering candidate capabilities To discover candidate capabilities, we abstract the low-level execution traces E in terms of the user's vocabulary (line 2). This abstraction of a low-level execution trace $\langle s_0, s_1, \ldots, s_{n-1}, s_n \rangle$ gives a high-level execution trace $\langle \tilde{s}_0, \tilde{s}_1, \ldots, \tilde{s}_{n-1}, \tilde{s}_n \rangle$. Since we do not assume that the user's vocabulary is precise enough to discern all the states available to the agent, more than one low-level state in an execution trace may be abstracted to a single high-level abstract state in $\tilde{S}$. Hence for some $j \in [0, n-1]$, it is possible that $\tilde{s}_j = \tilde{s}_{j+1}$. E.g., in Fig.1(a), the state available to the agent in the simulator expresses pixel-level details of the game (Fig.1(b)), whereas the user's vocabulary can express it only as an abstract state that represents multiple similar low-level states (Fig.1(c)). Formally, an *abstract execution trace* is the longest subsequence of $\tilde{s}_1, \ldots, \tilde{s}_n$ such that no two subsequent elements are identical. We remove the repetitions from the high-level execution trace to get the abstract execution trace $\tilde{e} = \langle \tilde{s}_0, \ldots, \tilde{s}_m \rangle$, where $m \leq n$.

We store each transition $\tilde{s}_i, \tilde{s}_{i+1}$ in $\tilde{e}$ as a new grounded candidate capability $\tilde{c}^*_{\tilde{s}_i, \tilde{s}_{i+1}}$.

Generating partial capability descriptions For each candidate capability $\tilde{c}^*_{\tilde{s}_i, \tilde{s}_{i+1}}$, the set of predicates $\tilde{s}_{i+1} \setminus \tilde{s}_i$ is added to the effects of $\tilde{c}^*_{\tilde{s}_i, \tilde{s}_{i+1}}$ in positive form (add effects); whereas the set $\tilde{s}_i \setminus \tilde{s}_{i+1}$ is added to the same candidate capability's effects in negative form (delete effects). As an optimization, in a manner similar to Stern and Juba (2017), we also store that the predicates true in $\tilde{s}_i$ cannot be negative preconditions for this capability, whereas the predicates false in $\tilde{s}_i$ cannot be positive preconditions.

Lifting the partial capability descriptions After line 2 of Alg. 1, we get a set of candidate capabilities with their partial descriptions that are in terms of predicates $\tilde{P}$ instantiated with objects in $\tilde{O}$. For each such grounded partial capability description, the predicates in the preconditions and effects are sorted in some lexicographic order. The choice of ordering is not important as long as it stays consistent throughout Alg. 1. The objects used in predicate arguments are assigned unique IDs corresponding to this capability in the order of their appearance in ordered predicates. These IDs are then used as variables representing capability parameters. E.g., suppose we have a grounded partial capability description with a precondition: *(alive ganon), (at link cell6), (at ganon cell5),(next_to ganon)*. Traversing the predicates in this order, the objects used in these predicates are given IDs as fol-

lows: {*ganon: 1, link: 2, cell6: 3, cell5: 4*}. Note that there is only one assignment per object, hence *ganon* in *(at ganon cell5)* was not given a separate ID. This procedure is extended to effects while assigning new IDs for any unseen objects in the partial capability description. Finally, the parameterized partial capability description is constructed by replacing all occurrences of objects in the partial capability description with variables corresponding to their unique IDs.

Combining candidate capabilities Multiple candidate partial capabilities can be combined if their precondition and effect conjunctions are unifiable. E.g., for any capability to match the capability discussed above, it's precondition should be in the form: *(alive ?1), (at ?2 ?3), (at ?1 ?4), (next_to ?1)*. Its effects should also be unifiable in terms of these parameters. The algorithm also keeps track of which grounded partial candidate capabilities map to each parameterized partial capability description. These descriptions are partial as they are generated using limited execution traces and may not capture all the preconditions or effects of a capability. E.g., suppose a capability adds a literal on its execution. If that literal is already present in the state where the capability was executed, it will not be captured in the effect of the capability's partial description. Hence, we next try to complete the partial capability descriptions. Note that all parameterized partial capability descriptions are collectively used as the parameterized capability model $\tilde{M}$ (line 4).

3.2 Completing Partial Capability Descriptions

To complete the partial capability descriptions $\tilde{M}$, Alg. 1 generates queries aimed to gain more information about the conditions under which the capability can be executed and the state properties that become true or false upon its execution. These queries give the agent a sequence of states, called waypoints, to traverse. Based on the agent's ability to traverse them, we derive the precondition and effect of each capability. Alg. 1 iterates through the combinations of predicates and capabilities generated earlier to determine how each predicate will appear in each capability's precondition and effect (line 6). For each combination, it generates a query as follows.

Active query generation For each combination of predicate, capability, and precondition (or effect), three possible capability descriptions $M_+, M_-, M_\emptyset$ are possible, one each for the predicate appearing in the precondition (or effect) of the capability in positive, negative, or absent mode, respectively (line 7). As noted when generating partial capability descriptions in Sec. 3.1, some of the models will not be considered if we know that a form is not possible for a particular predicate. The algorithm iteratively picks two such models M_x, M_y from $M_+, M_-, M_\emptyset$ (line 8) and generates a query $\tilde{q}$ in the form of a state $\tilde{s}_0$ and a capability sequence $\tilde{\pi}$ such that the result of executing the sequence $\tilde{\pi}$ on $\tilde{s}_0$ is different in M_x and M_y (line 9). We use the agent interrogation algorithm (AIA), from our prior work Verma, Marpally, and Srivastava (2021) (henceforth referred to as VMS21). For their process, AIA reduces query generation to a planning

problem. The resulting query $\tilde{q}$ is of the form $\langle \tilde{s}_0, \tilde{\pi} \rangle$, asking the model (or an agent) about the length of the plan $\tilde{\pi}$ that it can successfully execute when starting from state $\tilde{s}_0$. Here plan $\tilde{\pi}$ is a sequence of capabilities $\langle \tilde{c}_1^*, \ldots, \tilde{c}_n^* \rangle$ grounded with objects in $\tilde{O}$.

Generating waypoints from queries The queries described above cannot be directly posed to an agent, as the plan $\tilde{\pi}$ is in terms of high-level capabilities $\tilde{c}_i^* \in \tilde{C}^*$, which the agent will not be able to comprehend. Additionally, these high-level capabilities cannot be converted directly to low-level actions, as each capability may correspond to a different sequence of low-level actions depending on the state in which it is executed. Hence, we pose the queries to the agent in the form of high-level state transitions induced by the capabilities in the query's capability sequence.

To accomplish this, Alg. 1 converts the query $\tilde{q}$ to a sequence of waypoints $\tilde{\varrho} = \langle \tilde{s}_0, \ldots, \tilde{s}_n \rangle$. Starting from the initial state $\tilde{s}_0$, these are generated by applying the capability $\tilde{c}_i^*$, for $i \in [1, n]$, in the state $\tilde{s}_{i-1}$ according to the partial capability description of $\tilde{c}_i^*$. Note that the waypoints $\tilde{\varrho}$ cannot be presented to the agent as the agent may not know the high-level vocabulary. Hence these high-level waypoints must be refined into the low-level waypoints $\varrho = \langle s_0, \ldots, s_n \rangle$ (with each s_i similar to state shown in Fig. 1(b)) that agent understands.

Alg. 1 first converts the high-level waypoints $\tilde{\varrho}$ to a sequence of low-level waypoints $\varrho = \langle s_0, \ldots, s_n \rangle$ using the predicate definitions (line 11). Then each consecutive pair of states $\langle s_i, s_{i+1} \rangle$ is given sequentially to the agent as a *state reachability query* asking if it can reach from state s_i to s_{i+1} using its internal black-box policy (line 13).

Updating partial models based on agent responses Using its internal planning mechanism and the simulator, the agent attempts to reach from state s_i to s_{i+1}. If it succeeds, the response to the query is recorded as true; if it fails, the response is recorded as false. The algorithm keeps track of the waypoints that were successfully traversed. Based on the waypoint pairs that the agent was able to traverse, we discard the capability descriptions among M_x and M_y that are not consistent with the agent's response (line 15).

E.g., suppose the algorithm is trying to determine how the predicate *(alive ?monster1)* should appear in the precondition of capability $c4$ shown in Fig. 1(d). Now the two possible capability descriptions M_1 and M_2 that Alg. 1 is considering in line 6 are M_+ and M_-, corresponding to *(alive ?monster1)* being in $c4$'s precondition in positive and negative form, respectively. The algorithm will generate query with its corresponding waypoints $\tilde{\varrho} = \langle \tilde{s}_0, \tilde{s}_1 \rangle$, where $\tilde{s}_0$ will correspond to the state shown in Fig 1(a), and $\tilde{s}_1$ will be $\tilde{s}_0$ without Ganon. Now the agent uses its own internal mechanism to try to reach $\tilde{s}_1$ from $\tilde{s}_0$ and succeeds. Since this is not possible according to M_-, M_- will be discarded.

We now define and prove the theoretical properties of iCaML algorithm. To do this, we use two key properties of VMS21 relevant to this work: (1) if there exists a distinguishing query for two models then it will be generated (Thm. 1 in VMS21); and (2) the algorithm will not discard any model that is consistent with the agent (Thm. 2 in

VMS21). Interested readers can refer to VMS21 for further details.

3.3 Formal Analysis

Alg. 1 has two main desirable properties: (1) the partial capability model (that is maintained as $\tilde{M}$) is always maximally consistent, i.e, adding any more literals into it would be unsupported by the execution traces that we obtain; and (2) the final parameterized capability is complete in the limit of infinite execution traces given to Alg. 1. We first define these concepts and then formalize the results under Thm. 1 and Thm. 2.

Definition 5. Let $e = \langle s_0, \ldots, s_n \rangle$ be an execution trace with a corresponding abstract execution trace $\tilde{e} = \langle \tilde{s}_0, \ldots, \tilde{s}_m \rangle$, where $m \leq n$. A *parameterized capability model* $\tilde{M} = \langle \tilde{P}, \tilde{C}, \tilde{O} \rangle$ is consistent with E iff $\forall i \in [0, m-1]\ \exists \tilde{c}^* \in \tilde{C}^*\ \tilde{s}_{i+1} = \tilde{c}^*(\tilde{s}_i)$, where $\tilde{C}^*$ is a set of grounded capabilities that can be generated by instantiating the parameters of capabilities $\tilde{c} \in \tilde{C}$ with objects in $\tilde{O}$.

We extend this terminology to say that a capability model is consistent with a set of execution traces E iff it is consistent with every trace in E. This notion of consistency captures completeness as a parameterized capability model $\tilde{M}$ that is consistent with a set of execution traces E, is also complete w.r.t. E. We next define a stronger notion of completeness that our algorithm provides in the form of maximal consistency. This helps to assess the succinctness of a capability model with a set of execution traces E.

Definition 6. Let E be a set of execution traces, and Λ be the set of possible agents that can generate all execution traces in E. A *parameterized capability model* $\tilde{M} = \langle \tilde{P}, \tilde{C}, \tilde{O} \rangle$ *is maximally consistent with a set of execution traces E* iff (i) $\tilde{M}$ is consistent with E, and (ii) adding any predicate as positive or negative precondition or effect of a capability in $\tilde{M}$ makes it inconsistent with at least one execution trace that can be generated by at least one agent $\mathcal{A}^\# \in \Lambda$.

An abstraction satisfies *local connectivity* iff $\forall \tilde{s}\ \forall s_i, s_j \in f^{-1}(\tilde{s})$ there exists a sequence of primitive actions $\langle a_i, \ldots, a_n \rangle$ such that $a_n(a_{n-1} \ldots (a_1(s_i)) \ldots) = s_j$. We use this to show that the parameterized capability model learned by Alg. 1 is maximally consistent.

Theorem 1. *Let $\mathcal{A} = \langle S, A, T \rangle$ be an agent operating in a deterministic, fully observable, and stationary environment with a state space S using a set of primitive actions A. Given an input vocabulary $\tilde{P}$, and the set of execution traces E generated by $\mathcal{A}$, if local connectivity holds, then the capability model $\tilde{M}$ maintained by Alg. 1 is* consistent *with the set of execution traces E.*

Proof. We show that given the set of all execution traces E, the parameterized capability model $\tilde{M}$ maintained by Alg. 1 is consistent with E, i.e., for every high-level transition $\tilde{s}, \tilde{s}'$ corresponding to a transition in E, there exists a capability $\tilde{c}$ which has a grounding $\tilde{c}^*$ such that $\tilde{c}^*(s) = \tilde{s}'$. We prove this by contradiction. The partial capability model

$\tilde{M}$ is initially generated using observed transitions $\tilde{s}, \tilde{s}'$ corresponding to the transitions in E as grounded capabilities $\tilde{c}^*_{\tilde{s}, \tilde{s}'}$ (lines 2 to 4 in Alg. 1). So the model $\tilde{M}$ is consistent with the set of traces to start with. At each step, Alg. 1 adds a new literal l to a capability $\tilde{c}$ in $\tilde{M}$ such that adding l keeps $\tilde{M}$ consistent with the agent $\mathcal{A}$ (Thm. 2 from VMS21). Now consider that adding l to $\tilde{M}$ makes it inconsistent with an execution trace in E, i.e., there must exist a transition $\tilde{s}_1, \tilde{s}_2$ such that no capability $\tilde{c}^* \in \tilde{C}^*$ corresponds to it.

Consider the version of $\tilde{c}_1$ corresponding to $c^*_{\tilde{s}_1, \tilde{s}_2}$ that was modified by Alg. 1. We show that modifications inconsistent with this transition are not possible under the assumption that the agent's capabilities can be expressed using the input vocabulary.

Case 1: Suppose Alg. 1 added a literal l in the precondition of $\tilde{c}_1$ that was not true in $\tilde{s}_1$. Thm. 2 in VMS21 implies that absent and negated forms of l were inconsistent with executions of $\tilde{c}_1$ using the same agent that generated E. In other words, the agent sometimes requires l as a precondition to execute $\tilde{c}_1$, even though l was not a part of $\tilde{s}_1$. This contradicts the assumption that $\tilde{c}_1$ is expressible using the input vocabulary in the form of Def. 3.

Case 2: Suppose Alg. 1 added a literal l in the effect of $\tilde{c}_1$ that was not present in $\tilde{s}_2$. This implies that the negation and absence of l in the result of $\tilde{c}_1$ were inconsistent with the agent's execution of $\tilde{c}_1$ in query-responses generated by Alg. 1. A similar contradiction about the assumption of expressiveness follows.

Hence, the capability model $\tilde{M}$ maintained by Alg. 1 is *consistent* with the set of execution traces E. $\square$

Theorem 2. *Let $\mathcal{A} = \langle S, A, T \rangle$ be an agent operating in a deterministic, fully observable, and stationary environment with a state-space S using a set of primitive actions A. Given an input vocabulary $\tilde{P}$, and the set of execution traces E generated by $\mathcal{A}$, if local connectivity holds, then the capability model $\tilde{M}$ returned by Alg. 1 is* maximally consistent *with the set of execution traces E.*

Proof. We will prove the two conditions for maximal consistency separately. The first condition is that the model $\tilde{M}$ returned by Alg. 1 is consistent with E follows directly from Thm. 1. Since the model maintained by Alg. 1 at each step is consistent with E, hence the same model returned after the last iteration is also consistent with E.

Next, we show that adding any predicate as a positive or negative precondition or effect of a capability in $\tilde{M}$ returned by Alg. 1 makes it inconsistent with at least one execution trace that can be generated by at least one agent $\mathcal{A}^\# \in \Lambda$, where Λ is the set of possible agents that can generate all execution traces in E. We prove this by contradiction. Note that a literal is not added by Alg. 1 to an action's precondition (or effect) only if (1) in the observed traces, it was not present in the state where (immediately after) that action was executed; or (2) adding it in the precondition (or effect) of an action resulted in a response to a query that was inconsistent with that of the agent. Also, note that a predicate corresponding to a literal is always added to the model in some

form in each precondition (or effect). Suppose a literal l that was not added by Alg. 1 is added to $\tilde{M}$ in precondition (or effect) of a capability $\tilde{c}$ without making it inconsistent with the agent. Since a predicate p corresponding to this literal l is already present in $\tilde{c}$, this implies that the form of the predicate p added by Alg. 1 is incorrect. But this is not possible as shown by Thm. 1 and Thm. 2 of VMS21. Hence this is not possible and adding an additional literal in any form to an action's precondition or effect would make it inconsistent with the agent. This means that it also makes the model inconsistent with at least one agent $\mathcal{A}^\# \in \Lambda$. $\square$

Next, we formalize the notion of downward refinability, that the discovered capabilities are indeed within the agent's scope. In this work, refinability is similar to the notion of forall-exists abstractions (Srivastava, Russell, and Pinto 2016) for deterministic systems. Recall the notion of abstraction functions (Def. 4).

Definition 7. Let $\tilde{M} = \langle \tilde{P}, \tilde{C}, \tilde{O} \rangle$ be a capability model with $\tilde{S}$, the induced state space over $\tilde{P}, \tilde{O}$ using an abstraction function f, for an agent $\mathcal{A} = \langle S, A, T \rangle$; and $\tilde{C}^*$ be a set of grounded capabilities that can be generated by instantiating the arguments of capabilities $\tilde{c} \in \tilde{C}$ with objects in $\tilde{O}$. A capability $\tilde{c}^* \in \tilde{C}^*$ is *realizable w.r.t.* $\mathcal{A}$ iff $\forall \tilde{s} \in \tilde{S}$, if $\tilde{s} \models pre(\tilde{c}^*)$ then $\forall s \in f^{-1}(\tilde{s})$ $\exists a_1, \ldots, a_n \in A$: $a_n(a_{n-1} \ldots (a_1(s)) \ldots) \in \tilde{c}^*(\tilde{s})$. The model M is *realizable w.r.t.* $\mathcal{A}$ iff all capabilities $\tilde{c}^* \in \tilde{C}^*$ are realizable.

In these terms, discovered capabilities are more likely to be useful if they are accurate in the sense that they are consistent with execution traces and realizable, i.e., true representations of what the agent can do. Realizability captures the soundness of the model wrt the execution of the capabilities. We now show that the parameterized capability model that we learn is realizable.

Theorem 3. *Let $\tilde{P}$ be a set of predicates $\tilde{P}$, $\mathcal{A} = \langle S, A, T \rangle$ be an agent with a deterministic transition system T. If a high-level model is expressible deterministically using the predicates $\tilde{P}$, and local connectivity is ensured, then the parameterized capability model $\tilde{M}$ learned by Alg. 1 is* realizable*.*

Proof. We will prove that for all capabilities in $\tilde{C}$ learned as part of the parameterized capability model $\tilde{M}$, for all groundings $\tilde{C}^*$, if the capability is executed in an abstract state $\tilde{s}$ such that $\tilde{s} \models pre(\tilde{c}^*)$ then there exists a sequence of low-level states that the agent can traverse to reach a state $\tilde{s}' \in \tilde{c}^*(\tilde{s})$.

We prove this by cases. Consider a capability $\tilde{c} \in \tilde{C}$ whose description is learned using Alg. 1. Using Thm. 1, the precondition and effect of $\tilde{c}$ will be consistent with E generated by the agent. Now consider a grounded capability $\tilde{c}^*$ corresponding to the capability $\tilde{c}$. There are only two cases possible: (1) either $\tilde{c}^*$ appeared in the observed traces or was executed successfully by the agent in response to one of the queries posed to the agent; or (2) it was not present in either. We prove each case separately.

Case 1: There exists a set of low-level states s and s' such

that $\tilde{c}^*(\tilde{s}) = \tilde{s}'$, where $\tilde{s} = f(s)$ and $\tilde{s}' = f(s')$. Now due to local connectivity, all states in $f^{-1}(s)$ are connected with each other and same is true for all states in $f^{-1}(s')$. Hence the agent can traverse from any state in $f^{-1}(s)$ to any state in $f^{-1}(s')$ on executing the capability $\tilde{c}^*$. This makes the capability $\tilde{c}^*$ realizable.

Case 2: Since $\tilde{c}^*$ was not observed directly and the only way capabilities are added to $\tilde{M}$ is if they are lifted forms of capabilities identified from observation traces E, $\tilde{c}^*$ must be a grounding of the lifted form $\tilde{c}_1$ of a capability $\tilde{c}_1^*$ that is of the type considered in case 1. Alg. 1 constructs precondition and effect of $\tilde{c}_1$ while ensuring consistency with query responses and observations under the assumption that the capability model is expressible as in Def. 3. When this assumption holds, the effect or precondition of a capability can only depend on the vocabulary of available predicates, which are considered exhaustively (hierarchically) by Alg. 1. This implies that there must be a path from a concrete state s in the grounding corresponding to $\tilde{c}_1$'s precondition to a concrete state s' that satisfies the effects of grounding of $\tilde{c}_1$'s effects. By local connectivity, this extends to all concrete states in the same abstract state as $\tilde{s}$ corresponding to s.

Hence if a high-level model is expressible deterministically using the predicates $\tilde{P}$, and local connectivity is ensured, then the parameterized capability model $\tilde{M}$ learned by Alg. 1 is *realizable*. $\quad\square$

Note that here expressibility of a high-level model refers to the class of models of the form defined in Def. 3. Together, the notions of maximal consistency and realizability establish the completeness and soundness of our approach wrt a set of execution traces E. Note that this approach will also work when we have access to a stream of execution traces E being collected at random, independent of our active querying mechanism. We next show that in the limit of infinite randomly generated execution traces, our approach will capture all possible agent capabilities with probability 1. Here, capturing all possible agent capabilities in a learned model $\tilde{M} = \langle \tilde{P}, \tilde{C}, \tilde{O} \rangle$ means that if the agent can go from $\tilde{s}_i$ to $\tilde{s}_j$, then one of the capabilities in $\tilde{C}$ will be instantiable to result in $\tilde{s}_j$ when executed from $\tilde{s}_i$.

Theorem 4. *Let $\tilde{P}$ be a set of predicates, $\mathcal{A} = \langle S, A, T \rangle$ be an agent with a deterministic transition system T. Suppose random samples of agent behavior in the form of execution traces E are coming from a distribution that assigns non-zero probability to at least one transition corresponding to each ground capability ($\tilde{c}^*_{\tilde{s}_i, \tilde{s}_j}$, $\tilde{s}_i, \tilde{s}_j \subseteq \tilde{P}$). If a high-level model is expressible deterministically using the predicates $\tilde{P}$ and local connectivity holds, then in the limit of infinite execution traces E, the probability of discovering all capabilities $\tilde{c} \in \tilde{C}$ expressible using the predicates $\tilde{P}$ is 1.*

Proof. Consider every possible abstract transition that the agent can make. There are finite (let's consider L) such transitions possible given the predicate vocabulary $\tilde{P}$ and a fixed set of objects $\tilde{O}$. Now we are getting random execution traces E from a distribution that assigns non-zero probability to at least one transition corresponding to each
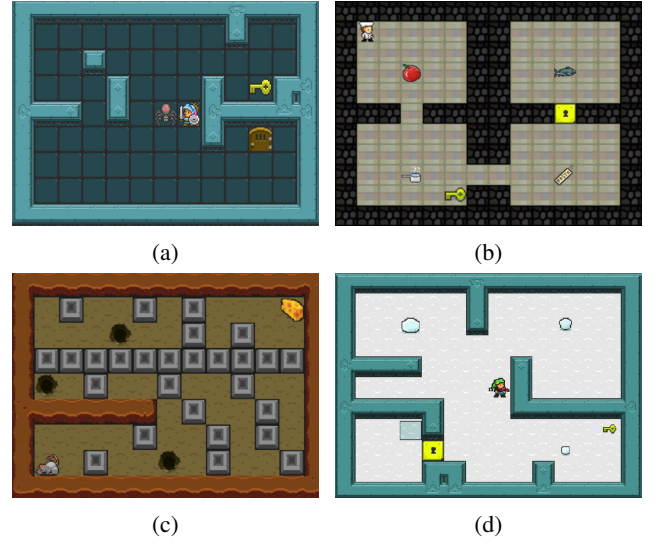

Figure 2: GVGAI's domains; (a) Zelda, (b) Cook-Me-Pasta, (c) Escape, and (d) Snowman.

ground capability ($\tilde{c}^*_{\tilde{s}_i, \tilde{s}_j}$). This means that the probability of not observing this finite set of cardinality L will reduce with each successive collection of L execution traces. Hence we will eventually observe at least one transition corresponding to each ground capability ($\tilde{c}^*_{\tilde{s}_i, \tilde{s}_j}$). Then as shown in Thm. 1, we will discover the capability $\tilde{c}$ corresponding to the ground transition $\tilde{c}^*_{\tilde{s}_i, \tilde{s}_j}$ with probability 1. $\quad\square$

4 Empirical Evaluation

We implemented Alg. 1 in Python to empirically verify its effectiveness.[1] To show that our approach can work with different internal agent implementations, we evaluated Alg. 1 with two broad categories of input test agents: *Policy agents* can use (possibly learned) black-box policies to plan and to respond to state reachability queries. We used policy agents with hand-coded policies for this evaluation. *Search agents* respond to the state reachability queries using arbitrary search algorithms. We used search agents that use A^* search (Hart, Nilsson, and Raphael 1968). We now describe the setup of our experiments used for evaluation.

4.1 Experimental Setup

Our test agents use the General Video Game Artificial Intelligence framework (Perez-Liebana et al. 2016; Perez-Liebana et al. 2019). Domains in GVGAI are two-dimensional ATARI-like games defined using the Video Game Description Language PyVGDL (Schaul 2013). We performed experiments on four such game domains – Zelda, Cook-Me-Pasta, Escape, and Snowman (Fig. 2). All these domains require the agent to navigate in a grid-based environment and complete a set of tasks (in some partial order) to complete the game. More details about these domains and the input user vocabularies are available in the extended version of the paper (Verma, Marpally, and Srivastava 2022).

Figure 3: 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 (number of cells in the grid) in the four GVGAI domains.

Since the complete list of an agent's capabilities may be irrelevant to a user's current needs, w.l.o.g, our implementation supports an input including 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 concept vocabulary. We also consider object types to be a subset of the unary predicates in the vocabulary and assume that each object has exactly one type. These types are used and discovered in capability like any other predicate. In addition, they are used in creating parameterized capability parameters as shown in Fig. 1(d).

For each domain, and for each grid size in that domain, we create a random game instance with the goal of achieving one of the user's specified properties of interest. To generate these instances, the number of obstacles in all domains, except Escape, 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 execution trace that is used in lines 1-2 of Alg. 1. These solutions are not always optimal. All experiments are run on 5.0 GHz Intel i9 CPUs with 64 GB RAM running Ubuntu 18.04.

As shown in Sec. 3.3, Alg. 1 is guaranteed to compute capability descriptions that are correct in the sense that they are consistent with the execution traces, and refinable and executable with respect to the true capabilities of the agent. We now present the main conclusions of our empirical analysis.

We evaluated our algorithm's performance along two aspects; (i) how the performance of our approach changes with respect to the size of the problem; and (ii) how its performance differs for search-based vs policy-based agents.

4.2 Empirical Results

Scalability analysis We increase the size of each domain to analyze its effect on the performance of the search and policy agents. Fig. 3 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*. The increasing number of queries is an expected
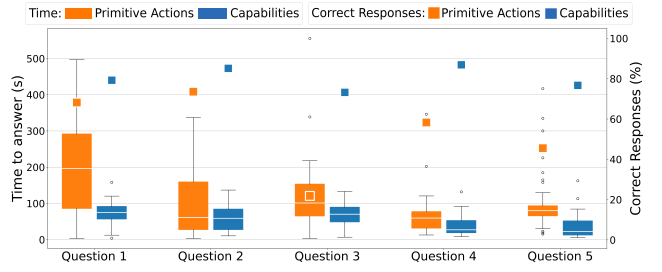

Figure 4: Data from behavior analysis shows that using computed capability descriptions took lesser time and yielded more accurate results. See Sec. 4.3 for details.

behavior and this is also clear in approaches that use passive observations of agent behavior (Yang, Wu, and Jiang 2007; Aineto, Celorrio, and Onaindia 2019).

Agent type analysis The *number of queries required by the policy agent is higher than that of the search agent* in almost all cases. This is because a large number of state reachability queries fail on the policy agent as the sequence of waypoints in these queries 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 state reachability queries by following their policy, whereas the search agents perform an exhaustive search of the state space for every such query.

4.3 User Study

We conducted a user study to evaluate the utility of the capability descriptions discovered and computed by Alg. 1. Intuitively, our notion of interpretability matches that of common English and its use in AI literature, e.g., as enunciated by Doshi-Velez and Kim (2018): *"the ability to explain or to present in understandable terms to a human."* We evaluate this through the following operational hypothesis:

H1. The discovered capabilities make it easier for users to analyze and predict the outcome of the agent's possible behaviors.

We designed the following study to evaluate H1.

Behavior analysis study This study compares the predictability and analyzability of agent behavior in terms of the agent's low-level actions and high-level capabilities. Each user is explained the rules of a Zelda-like game. One group of users – called the *primitive action group* – are presented with text descriptions of the agent's primitive actions, while the users in the other group – called the *capability group* – are presented with a text description of the six learned capabilities. The capability group users are asked to choose a short summarization for each capability description, out of the eight possible summarizations that we provide, whereas the primitive action group users are asked to choose a short summarization for each primitive action description, out of the five possible summarizations that we provide. Then each user is given the same 5 questions in order. Each question contains two game-state images; start and end state. The user is asked what sequence of actions or capabilities that

the agent should execute to reach the end state from the start state. Each question has 5 possible options for the user to choose from, and these options differ depending on their group. We then collect the data about the accuracy of the answers, and the time taken to answer each question.

Study design   108 participants were recruited from Amazon Mechanical Turk and randomly divided into two groups of 54 each. Each user was provided with a survey on Qualtrics (Qualtrics 2005) that explained the rules of GV-GAI's Zelda game. We used screeners (Kennedy et al. 2020; Arndt et al. 2021) to ensure quality of the data collected, and discarded 23 responses. The results are based on the responses of 41 and 43 users in the primitive action and capability group, respectively.

Results   The results of the behavior analysis study are shown in (Fig. 4) To evaluate the statistical significance (p-value) of the difference in the mean of the time taken by the two groups, we used Student's t-test (Student 1908). The results indicate that the test results were statistically significant with p-values less than 0.05 for all five questions. Also, the users took less time to answer questions and they got more responses correct when using the capabilities as compared to using primitive actions. This validates H1 that the discovered capabilities made it easier for the users to analyze and predict the agent's behavior correctly. Detailed information about the user study is available in the extended version of the paper (Verma, Marpally, and Srivastava 2022).

5   Related Work

High-level skills from input options   Given a set of options encoding skills as input, Konidaris, Kaelbling, and Lozano-Perez (2018) and James, Rosman, and Konidaris (2020) propose methods for learning high-level propositional models of options representing various "skills." They assume access to predefined options and learn the high-level symbols that describe those options at the high-level. 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 apriori inputs to learn agent capabilities: high-level actions as well as their interpretable descriptions in terms of the input vocabulary.

Learning symbolic models using physics simulators   Multiple approaches learn different kinds of symbolic models of the functionality of ATARI or physics-based simulators using methods like conjunctions of binary input features (Kansky et al. 2017), graph neural networks (Battaglia et al. 2016; Cranmer et al. 2020), CNNs (Agrawal et al. 2016; Fragkiadaki et al. 2016), etc. Some methods create interpretable descriptions of reinforcement learning policies using trees (Liu et al. 2018) or specialized programming languages (Verma et al. 2018). These approaches solve the orthogonal problem of learning the functionality of an agent that could help a user understand how an agent would solve a problem, whereas we focus on learning capabilities of the

agent that could help a user understand and answer what type of problems it could solve.

Action model learning   The planning community has also worked on learning STRIPS-like action models of agent functionality from observations of its behavior (Gil 1994; Yang, Wu, and Jiang 2007; Cresswell, McCluskey, and West 2009; Zhuo and Kambhampati 2013; Stern and Juba 2017; Aineto, Celorrio, and Onaindia 2019; Bonet and Geffner 2020). Jiménez et al. (2012) and Arora et al. (2018) present a comprehensive review of such approaches. These methods work with broad assumptions that the agent model is internally expressed in the same vocabulary as the user's (Gil 1994; Weber, Morwood, and Bryce 2011; Juba, Le, and Stern 2021), or at a similar level of abstraction (Mehta, Tadepalli, and Fern 2011; Verma, Marpally, and Srivastava 2021; Nayyar, Verma, and Srivastava 2022). Additionally, such methods have as input a given set of predicates in terms of which they learn the functionality descriptions of the agent.

High-level actions   Works like Madumal et al. (2020) explain an agent's policy in terms of high-level actions but they assume that high-level actions are a part of the input whereas our approach discovers these actions. There is an orthogonal thread of research on using high-level actions in AI planning as tasks, and learning low-level policies for each of those tasks (Yang et al. 2018; Lyu et al. 2019; Illanes et al. 2020; Kokel et al. 2021). These works assume the high-level actions as input and learn the corresponding low-level policies.

As compared to the above two classes of methods, our work focuses on solving the harder problem of discovering the capabilities of the agent behavior resulting from its planning/learning algorithms and learning the descriptions of these capabilities.

6   Conclusion

We presented a novel approach for learning the capability description of an AI system in terms of user-interpretable concepts by combining information from passive execution traces and 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 for the agents that internally use black-box deterministic policies, or search techniques, we can successfully discover the capabilities and their descriptions. Extending this approach for partially observable settings and relaxing the various assumptions we made are some of the promising future directions for this work.

Acknowledgements

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*.

Aineto, D.; Celorrio, S. J.; and Onaindia, E. 2019. Learning Action Models With Minimal Observability. *Artificial Intelligence* 275:104–137.

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

Arndt, A. D.; Ford, J. B.; Babin, B. J.; and Luong, V. 2021. Collecting Samples from Online Services: How to Use Screeners to Improve Data Quality. *International Journal of Research in Marketing*.

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*.

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*.

Bonet, B., and Geffner, H. 2020. Learning First-Order Symbolic Representations for Planning from the Structure of the State Space. In *Proc. ECAI*.

Camacho, A., and McIlraith, S. A. 2019. Learning interpretable models expressed in linear temporal logic. In *Proc. ICAPS*.

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*.

Cresswell, S.; McCluskey, T.; and West, M. 2009. Acquisition of Object-Centred Domain Models from Planning Examples. In *Proc. ICAPS*.

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*.

Doshi-Velez, F., and Kim, B. 2018. *Considerations for Evaluation and Generalization in Interpretable Machine Learning*. Springer International Publishing. 3–17.

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.

Greydanus, S.; Koul, A.; Dodge, J.; and Fern, A. 2018. Visualizing and Understanding Atari Agents. In *Proc. ICML*.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

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

Illanes, L.; Yan, X.; Icarte, R. T.; and McIlraith, S. A. 2020. Symbolic Plans as High-Level Instructions for Reinforcement Learning. 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.

Juba, B.; Le, H. S.; and Stern, R. 2021. Safe Learning of Lifted Action Models. In *Proc. KR*.

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*.

Kennedy, R.; Clifford, S.; Burleigh, T.; Waggoner, P. D.; Jewell, R.; and Winter, N. J. G. 2020. The Shape of and Solutions to the MTurk Quality Crisis. *Political Science Research and Methods* 8(4):614–629.

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

Kokel, H.; Manoharan, A.; Natarajan, S.; Ravindran, B.; and Tadepalli, P. 2021. RePReL: Integrating Relational Planning and Reinforcement Learning for Effective Abstraction. In *Proc. ICAPS*.

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.

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

Lyu, D.; Yang, F.; Liu, B.; and Gustafson, S. 2019. SDRL: Interpretable and Data-Efficient Deep Reinforcement Learning Leveraging Symbolic Planning. In *Proc. AAAI*.

Madumal, P.; Miller, T.; Sonenberg, L.; and Vetere, F. 2020. Explainable Reinforcement Learning Through a Causal Lens. In *Proc. AAAI*.

Malle, B. F. 2004. *How the Mind Explains Behavior: Folk Explanations, Meaning, and Social Interaction*. The MIT Press.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D. S.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Comp. Vision and Control.

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

Miller, T. 2019. Explanation in Artificial Intelligence: Insights from the Social Sciences. *Artificial Intelligence* 267:1–38.

Mou, Y., and Xu, K. 2017. The Media Inequality: Comparing the Initial Human-Human and Human-AI Social Interactions. *Computers in Human Behavior* 72:432–440.

Nayyar, R. K.; Verma, P.; and Srivastava, S. 2022. Differential Assessment of Black-Box AI Agents. In *Proc. AAAI*.

Paulus, R.; Xiong, C.; and Socher, R. 2018. A Deep Reinforced Model for Abstractive Summarization. In *Proc. ICML*.

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

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.

Popov, I.; Heess, N.; Lillicrap, T.; Hafner, R.; Barth-Maron, G.; Vecerik, M.; Lampe, T.; Tassa, Y.; Erez, T.; and Riedmiller, M. 2017. Data-efficient Deep Reinforcement Learning for Dexterous Manipulation. *arXiv preprint arXiv:1704.03073*.

Qualtrics. 2005. Qualtrics XM. https://www.qualtrics.com/. Accessed: 2022-05-10.

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

Russell, S. J. 1997. Rationality and Intelligence. *Artificial Intelligence* 94(1-2):57–77.

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

Schaul, T. 2013. A Video Game Description Language for Model-based or Interactive Learning. In *2013 IEEE Conference on Computational Intelligence in Games (CIG)*.

Sreedharan, S.; Soni, U.; Verma, M.; Srivastava, S.; and Kambhampati, S. 2022. Bridging the Gap: Providing Post-Hoc Symbolic Explanations for Sequential Decision-Making Problems with Inscrutable Representations. In *Proc. ICLR*.

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*.

Student. 1908. The Probable Error of a Mean. *Biometrika* 6(1):1–25.

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.; Marpally, S. R.; and Srivastava, S. 2022. Discovering User-Interpretable Capabilities of Black-Box Planning Agents. *CoRR* abs/2107.13668.

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

Yang, F.; Lyu, D.; Liu, B.; and Gustafson, S. 2018. Peorl: Integrating Symbolic Planning and Hierarchical Reinforcement Learning for Robust Decision-Making. In *Proc. IJCAI*.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning Action Models from Plan Examples Using Weighted MAX-SAT. *Artificial Intelligence* 171(2-3):107–143.

Zhuo, H. H., and Kambhampati, S. 2013. Action-Model Acquisition from Noisy Plan Traces. In *Proc. IJCAI*.