Learning Generalized Models by Interrogating Black-Box Autonomous Agents

Anonymized

Abstract

This paper develops a new approach for estimating the internal model of an autonomous agent that can plan and act, by interrogating it. In this approach, the user may ask an autonomous agent a series of questions, which the agent answers truthfully. Our main contribution is an algorithm that generates an interrogation policy in the form of a sequence of questions to be posed to the agent. Answers to these questions are used to derive a minimal, functionally indistinguishable class of agent models. While asking questions exhaustively for every aspect of the model can be infeasible even for small models, our approach generates questions in a hierarchical fashion to eliminate large classes of models that are inconsistent with the agent. Empirical evaluation of our approach shows that for a class of agents that may use arbitrary black-box transition systems for planning, our approach correctly and efficiently computes STRIPS-like agent models through this interrogation process.

1 Introduction

Growing deployment of autonomous agents ranging from personal digital assistants to self-driving cars leads to a pervasive problem: how would we ascertain whether an autonomous agent will be safe, reliable, or useful in a given situation? This problem becomes particularly challenging when we consider that most autonomous systems are not designed by their users; their internal software may be unavailable or difficult to understand; and they may adapt and learn from the environment where they are deployed, invalidating design-stage knowledge of agent models.

This paper presents a new approach for addressing this problem by estimating the internal model of a black-box autonomous agent, essentially by interrogating it. Consider a situation where a researcher wants a robot to clean up their chemistry lab. The robot has been delivered recently, and the researcher is unsure whether it correctly understands the task. If the lab assistant was a person, the researcher would have proceeded by asking a series of questions, e.g. "What do you think will happen if you picked up bottle 1 followed by bottle 2 and bottle 3 without putting down anything in between?" Answers to such questions help the researcher develop an understanding of the assistant's frame of knowledge, or "model", while placing a minimal introspective requirement on the agent. Most simulator-based and analytical-model based agents can easily answer such questions. However, generating the right set of questions to ask the agent in order to efficiently estimate the agent's current model is a challenging problem. The focus of this new direction of research is on solving this problem.

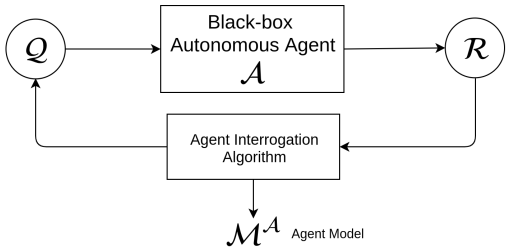

Figure 1: The agent interrogation framework

Although a number of approaches are being explored for learning agent models from available data (Stern and Juba 2017; Cresswell and Gregory 2011; Aineto et al. 2019) (see Sec. 2 for a survey), to the best of our knowledge this is the first approach addressing the problem of determining an agent's model in the absence of prior behavioral data, by generating questions and querying it.

In developing the first steps towards this paradigm, we assume that the user wishes to estimate the agent's internal model in the form of a STRIPS-like domain model with conjunctive preconditions, add lists, and delete lists (and that the agent's model is expressible as such), although our framework can be extended to handle other types of formal domain representations and we treat the agent as a black-box: it can use its internal model for simulations but it does not have access to the model's representation (Fig. 1). So we could not ask it for instance, "What are the preconditions of action A?" Further, we assume that the agent has functional definitions for the propositions and actions present in the user's vocabulary (these definitions can be programmed as Boolean functions over the state), and that it always answers truthfully. The problem of ascertaining a robot's model is challenging in this setting because the space of possible

models is typically too large to examine using naive approaches. E.g., three actions and five atomic propositions yield $\sim 10^{14}$ possible models. We make minimal assumptions about the agent's question-answering capabilities: we assume that it can answer queries about hypothetical executions, such as "what do you believe would happen if you did X", and "can you do X?", where X is a sequence of actions.

Our novel approach to this problem uses a top-down process that eliminates large classes of agent-inconsistent models by computing queries that discriminate between pairs of *abstract models*. When an abstract model's answer to a query differs from the agent's answer, we effectively eliminate the entire set of possible concrete models that are refinements of this abstract model. Our results show that the resulting approach efficiently estimates agent models in problems that are much larger than the three-action, five-proposition case highlighted above.

The rest of this paper is structured as follows. The following section presents a summary of related work; Section 3 gives the formal overview of our approach and the background terminology used in this paper, and also explains the approach that we have devised; Section 4 discusses the empirical evaluations of our approach; and Section 5 highlights the conclusions mentioning the future direction of this approach.

(a) $\mathcal{M}^{\mathcal{A}}$'s *pick* actions (unknown to $\mathcal{H}$)

*pick-b*1 action:

(handempty), (on-floor-b1)	→	(in-hand-b1), (¬(handempty)), (¬(on-floor-b1)

*pick-b*2 action:

(handempty), (on-floor-b2)	→	(in-hand-b2), (¬(handempty)), (¬(on-floor-b2)

(b) $\mathcal{M}_1$'s *pick* actions

*pick-b*1 action:

(handempty), (on-floor-b1)	→	(¬(handempty)), (¬(on-floor-b1)

*pick-b*2 action:

(handempty), (on-floor-b2)	→	(¬(handempty)), (¬(on-floor-b2)

(c) $\mathcal{M}_2$'s *pick* actions

*pick-b*1 action:

(on-floor-b1)	→	(¬(on-floor-b1)

*pick-b*2 action:

(on-floor-b2)	→	(¬(on-floor-b2)

Figure 2: *pick* actions of the agent model $\mathcal{M}^{\mathcal{A}}$ and two abstracted models $\mathcal{M}_1$ and $\mathcal{M}_2$. Here $X \rightarrow Y$ means that X is the precondition of an action and Y is the effect.

2 Related Work

Our solution approach takes motivation from the action model learning which has been used extensively to learn the agent model by analyzing the actions of the agent (Alterman 1986; Yang, Wu, and Jiang 2007; Stern and Juba 2017; Aineto, Jiménez, and Onaindia 2018). Our approach differs from these methods on the basis of how the plan traces are generated. These methods rely on plan traces generated apriori and hence limit the solution of model estimation to a set of equivalent models that cannot be refined further based on the information provided by plan traces.

LOCM (Cresswell, McCluskey, and West 2009) and LOCM2 (Cresswell and Gregory 2011) represent another class of algorithms that use finite state machines to create action models using plan traces. In contrast, we address the problem of querying the agent to infer its model in the absence of prior training data.

Recently the idea of reducing model recognition to a planning problem was explored (Aineto et al. 2019) under the assumption that a set of partially observable plan executions were available along with a set of possible models. This approach is shown to work even for unobservable intermediate actions and states.

But these methods also do not discuss how plan traces are generated and how much additional data would be needed to ensure the desired level of success in estimating the model. So the problem of choosing what data to collect is not addressed in these prevalent techniques. We address the problem of selecting the data to collect through query generation.

Approaches such as (Khardon and Roth 1996) address the orthogonal problem of making model-based inference faster *given a set of queries*, under the assumption that a static set of models represents the true knowledge base. In contrast, the focus of this paper is on incrementally generating queries that would reduce an initially inconsistent set of agent models to the true agent model.

3 Our Approach

Consider a case where we have a black-box autonomous agent $\mathcal{A}$. A human interrogator $\mathcal{H}$'s task is to compute a planning model $\mathcal{M}^{\mathcal{A}}$ that captures $\mathcal{A}$'s internal model. The only information $\mathcal{H}$ has is the set of actions $\mathbb{A}$ that $\mathcal{A}$ can perform. Since $\mathcal{A}$ has functional definitions of the predicates in $\mathcal{H}$'s vocabulary, i.e. $\mathcal{A}$ and $\mathcal{H}$ agree on the propositions $\mathbb{P}$, there is sufficient information for a dialog between $\mathcal{H}$ and $\mathcal{A}$ about the outcomes and executability of hypothetical action sequences. We will utilize a propositional representation for ease of exposition in most of this paper. However, our approach extends naturally to relational representations and our empirical evaluation uses a relational representation. In the absence of any other information, the number of possible models in a propositional representation is $9^{|\mathbb{P}|\times|\mathbb{A}|}$. This is because using one proposition and one action, we can generate 9 different models.

Our approach iteratively generate pairs of abstract models and eliminates one of them by asking $\mathcal{A}$ queries and comparing its answer with the answer generated using the abstract models.

Example 1. *Consider the case of the chemistry lab robot discussed in the introduction. Assume the researcher is considering two abstract models $\mathcal{M}_1$ and $\mathcal{M}_2$ having only the propositions $handempty$, $on\text{-}floor\text{-}b1$ and $on\text{-}floor\text{-}b2$ and the agent's model is $\mathcal{M}^{\mathcal{A}}$ (Fig. 2). $\mathcal{H}$ can ask the agent*

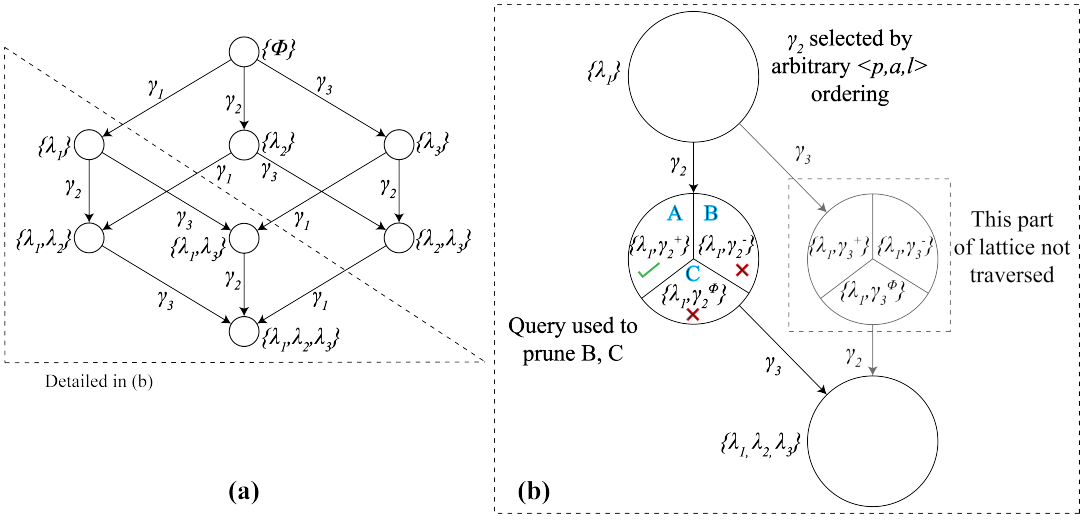

Figure 3: (a) A subset lattice created using pal-tuples; (b) Detailed view of a portion of lattice (marked in (a)) illustrating how partitions are created and pruned.

what will happen if $\mathcal{A}$ picks up multiple bottles one after another. The agent would respond that it cannot pick up the second bottle (due to the precondition handempty). Thus, we can eliminate the abstract model $\mathcal{M}_2$, which does not have the literal handempty as a precondition in action pick-b1 and pick-b2, and thus entails that picking up multiple bottles is possible.

The key to utilizing this approach is to be able to generate the right sequence of questions to ask the agent. Our approach continually generates the next question to ask until the set of possible models becomes functionally equivalent and cannot be reduced further.

3.1 Background

We assume that $\mathcal{H}$ needs to estimate $\mathcal{A}$'s model using a STRIPS like planning model (Fikes and Nilsson 1971) represented as a pair $\mathcal{M} = \langle \mathbb{P}, \mathbb{A} \rangle$, where $\mathbb{P} = \{p_1, \ldots, p_n\}$ is a finite set of propositions, each with a binary domain $dom(p_i)$ associated with it; $\mathbb{A} = \{a_1, \ldots, a_k\}$ is a finite set of actions, each represented as a tuple $\langle pre(a_j), eff(a_j) \rangle$. For each action a_j, *pre*(a_j) and *eff*(a_j) represent conjunctive sets of literals. In this representation, states are represented as (conjunctive) sets s of true propositions. Let *eff*$^+(a)$ (*eff*$^-(a)$) denote the positive (negative) literals in *eff*(a). The effect of an action a on a state $s \in 2^{\mathbb{P}}$ is represented as $a(s) = (s \cup \textit{eff}^+(a)) \setminus \textit{eff}^-(a)$ if *pre*$(a) \subseteq s$; $a(s)$ is undefined otherwise. A planning problem is defined as a 3-tuple $\langle \mathcal{M}, s_{\mathcal{I}}, S_{\mathcal{G}} \rangle$ where $\mathcal{M} = \langle \mathbb{P}, \mathbb{A} \rangle$ is the planning model, $s_{\mathcal{I}} \in 2^{\mathbb{P}}$ is the initial state, $S_{\mathcal{G}} \subseteq 2^{\mathbb{P}}$ is the goal condition. A plan π of length k is a sequence of actions $\langle a_1, \ldots, a_k \rangle$ that solves this planning problem if $s_{\mathcal{G}} \subseteq \pi(s_{\mathcal{I}})$. Here $\pi(s) = a_k(a_{k-1} \ldots (a_1(s)))$. This representation can be extended to the case of disjunctive preconditions, which can be converted to the disjunctive normal form (DNF) and each conjunctive sub-formula in this DNF can have a different action.

3.2 The Agent-Interrogation Task

We define the overall problem of agent interrogation as follows. Given a class of queries and an agent with an unknown model who can answer these queries, determine the model of the agent. Formally:

Definition 2. *An agent interrogation task is defined using a tuple $\langle \mathcal{M}^{\mathcal{A}}, \mathbb{Q} \rangle$ where $\mathcal{M}^{\mathcal{A}}$ is the model of the agent (unknown to the interrogator) being interrogated, and $\mathbb{Q}$ is the class of queries that can be posed to the agent by the interrogator.*

Let Θ be the set of possible answers to queries. Thus, strings $\theta^* \in \Theta^*$ denote the information received by $\mathcal{H}$ at any point in the query process. *Solutions to the agent interrogation task* take the form of a query policy $\theta^* \rightarrow \mathbb{Q} \cup \{\textit{Stop}\}$ that maps sequences of answers to the next query that the interrogator should ask. The process stops with the *Stop* query. In other words, all valid query policies map all sequences $x\theta$ to *Stop* whenever $x \in \Theta^*$ is mapped to *Stop*, for all answers $\theta \in \Theta$.

In order to formulate our solution approach, we consider a model $\mathcal{M}$ to be comprised of components called *palm* tuples of the form $\lambda = \langle p, a, l, m \rangle$ where $p \in \mathbb{P}$, $a \in \mathbb{A}$, $l \in \{\textit{pre}, \textit{eff}\}$ and $m \in \{+, -, \phi\}$. For convenience, we use subscripts p, a, l or m to denote the corresponding component in a palm tuple. The presence of a palm tuple λ in a model denotes the fact that in that model, the proposition λ_p appears in an action λ_a at a location λ_l as a true (false) literal when sign λ_m is positive (negative), and is absent when $\lambda_m = \phi$. This allows us to define the set-minus operation $M \setminus \lambda$ on this model as removing the palm-tuple λ from the model.

We consider two palm tuples $\lambda_1 = \langle p_1, a_1, l_1, m_1 \rangle$ and $\lambda_2 = \langle p_2, a_2, l_2, m_2 \rangle$ to be *variants* of each other ($\lambda_1 \sim \lambda_2$) iff they differ only on m, i.e. $\lambda_1 \sim \lambda_2 \Leftrightarrow (\lambda_{1_p} = \lambda_{2_p}) \wedge (\lambda_{1_a} = \lambda_{2_a}) \wedge (\lambda_{1_l} = \lambda_{2_l}) \wedge (\lambda_{1_m} \neq \lambda_{2_m})$.

Hence mode assignments to a *pal* tuple $\gamma = \langle p, a, l \rangle$ can

result in 3 palm variants $\gamma^+ = \langle p, a, l, + \rangle, \gamma^- = \langle p, a, l, - \rangle$, and $\gamma^\phi = \langle p, a, l, \phi \rangle$.

We are now ready to define the notion of abstractions used in our solution approach. Several approaches have explored the use of abstraction in planning (Sacerdoti 1974; Giunchiglia and Walsh 1992; Helmert et al. 2017; Bäckström and Jonsson 2013; Srivastava, Russell, and Pinto 2016). Def. 3 presents a special case of propositional abstractions because in propositional abstraction, we remove a proposition p from all actions and from all locations, but in our approach, we only remove a single λ tuple to create an abstraction.

Definition 3. *Let $\mathcal{U}$ be the set of all possible models. The **abstraction of a model** $\mathcal{M}$, on the basis of a palm tuple λ, is given by $f_\lambda : \mathcal{M} \mapsto (\mathcal{M} \setminus \lambda)$, where $f_\lambda : \mathcal{U} \to \mathcal{U}$. A set X is said to be a model abstraction of a set of models M with respect to a λ-tuple, if $X = \{f_\lambda(m) : m \in M\}$.*

We also use the notation $\mathcal{M}' \sqsubset_\lambda \mathcal{M}$ to represent the situation where $f_\lambda(\mathcal{M}) = \mathcal{M}'$.

We use this abstraction framework to define a subset-lattice over abstract models (Fig. 3(a)). Note that at each node we can have all possible variants of a palm tuple. For example the topemost node in fig 3(b), we can have models corresponding to γ_1^+, γ_1^-, and γ_1^ϕ. Each node in the lattice represents a collection of possible abstract models at the same level of abstraction. As we move up in the lattice, we get more abstracted version of the models and we get more concretized models as we move down.

Definition 4. *A **model lattice** $\mathcal{L}$ is a 5-tuple $\mathcal{L} = \langle N, E, \Gamma, \ell_N, \ell_E \rangle$, where N is a set of lattice nodes, Γ is the set of all pal tuples $\langle p, a, l \rangle$, $\ell_N : N \to 2^{2^\Lambda}$ is a node label function where $\Lambda = \Gamma \times \{+, -, \phi\}$, E is the set of lattice edges, and $\ell_E : E \to \Gamma$ is a function mapping edges to edge labels such that for each edge $n_i \to n_j, \ell_N(n_j) = \{\Lambda \cup \{\gamma^k\} | \Lambda \in \ell_N(n_i), \gamma = \ell_E(n_i \to n_j), k \in \{+, -, \phi\}\}$.*

The supremum $\top$ of the lattice $\mathcal{L}$ is the most abstracted node of the lattice, whereas the infimum $\bot$ is the most concretized node. Also, a node $n \in N$ in this lattice $\mathcal{L}$ can be uniquely identified as the sequence of pal tuples that label edges leading to it from the supremum. As shown in fig 3(b), even though theoretically $\ell : n \mapsto 2^{2^\Lambda}$, only one of the sets is stored at each node as the others are pruned out based on $\mathcal{Q}$. Also, in these model lattices every node has an edge going out of it corresponding to each pal tuple that is not present in the paths leading to it from the most abstracted node. At any stage in the interrogation process, we will use nodes in such a lattice to represent the set of models that are possible given the agent's responses up to that point. At every step, our query-generation algorithm will create queries that help us determine the next descending edge to take from each lattice node.

Query Classes As mentioned earlier, we pose queries to the agent and based on the responses we try to infer the agent's model. Formally, we express queries as functions mapping models to answers.

Definition 5. *Let $\mathcal{U}$ be the set of possible models and $\mathcal{R}$ a set of possible responses. A **query** $\mathcal{Q}$ is a function $\mathcal{Q} : \mathcal{U} \to \mathcal{R}$.*

In this paper we utilize two specific type of these queries: *plan result* queries and *plan executability* queries. Both types of queries are parameterized by a state s_I and a plan π.

Plan result queries ($\mathcal{Q}_R$) ask the agent for the state it will end up in if it executes the plan π when starting in the state $s_\mathcal{I} \in 2^\mathbb{P}$. E.g., "Given that bottles *b1* and *b2* are on the floor and your hand is empty, what do you think will happen after you pick Bottles *b1* and *b2* and keep them on shelf *s1*?" A response to this query can be of the form "*b1* is on shelf *s1* and *b2* is on shelf *s1*".

Formally, the response $\mathcal{R}_R$ for these queries is a tuple $\langle \ell, s_\mathcal{F} \rangle$, where ℓ is the number of steps for which the plan π was successfully able to run, and $s_\mathcal{F} \in 2^\mathbb{P}$ is the final state of the agent after executing the query. If the plan π is not executable according to the agent model $\mathcal{M}^\mathcal{A}$ then $\ell < len(\pi)$, otherwise if π is executable then $\ell = len(\pi)$ and $s_\mathcal{F} \in 2^\mathbb{P}$ such that $\mathcal{M}^\mathcal{A} \models \pi(s_\mathcal{I}) = s_\mathcal{F}$. Thus, $\mathcal{Q}_R : \mathcal{U} \to \mathbb{N} \times 2^\mathbb{P}$, where $\mathbb{N}$ is set of natural numbers.

Plan executability queries ($\mathcal{Q}_E$) ask the agent the length of the longest prefix of the plan π that can be executed successfully when starting in the state $s_\mathcal{I} \in 2^\mathbb{P}$. Responses to such queries include an error report indicating the state properties that would need to be changed in order for the agent to continue execution. E.g., "Given that the bottles *b1* and *b2* with labels *l1* and *l2* respectively are kept on the floor and your hand is empty, can you pick them up and place in shelf *s1* with label *l1*?" A response to this query can be of the form "I can execute this plan only till step 3. Bottle *b2* and shelf *s1* need to have the same label for the subsequent placement action." Formally, the response $\mathcal{R}_E$ for these queries is a tuple $\langle \ell, a_{Fail}, p_{Fail} \rangle$ where ℓ is the number of steps for which the plan π was successfully able to run, a_{Fail} is the first action that failed to execute, and p_{Fail} is a literal in a_{Fail}'s precondition that would not be satisfied at step $\ell < len(\pi)$. Essentially a_{Fail} and p_{Fail} are like error codes. Thus, $\mathcal{Q}_E : \mathcal{M} \to \mathbb{N} \times \mathbb{A} \times \mathbb{P}$, where $\mathbb{N}$ is the set of natural numbers.

Qualitatively not all queries are equivalent, some of them might not increase our knowledge of the agent model at all. Hence we define some properties associated with each query to ascertain its usability. A query is useful only if it can distinguish between two models.

Definition 6. *A query $\mathcal{Q}$ is said to **distinguish** a pair of models $\mathcal{M}_i$ and $\mathcal{M}_j$, denoted as $\mathcal{M}_i \rfloor^\mathcal{Q} \mathcal{M}_j$, iff $\mathcal{Q}(\mathcal{M}_i) \neq \mathcal{Q}(\mathcal{M}_j)$.*

For a given set of models we are concerned about two properties in particular. The first one is distinguishability which tells us whether a query can distinguish between two different abstract models based on their responses.

Definition 7. *Two models $\mathcal{M}_i$ and $\mathcal{M}_j$ are **distinguishable**, denoted as $\mathcal{M}_i \rfloor \mathcal{M}_j$, iff there exists a query that can distinguish between them, i.e. $\exists \mathcal{Q}\ \mathcal{M}_i \rfloor^\mathcal{Q} \mathcal{M}_j$.*

The second important property for a pair of models is prunability which tells us whether we can discard one of those models based on their response and the agent's response. However, the agent's response might be at different level of abstraction. When comparing the responses of

two models at different levels of abstraction, we also need to evaluate if the response of abstracted model $\mathcal{M}'$ is consistent with that of the agent, i.e. $\mathcal{Q}(\mathcal{M}^{\mathcal{A}}) \models \mathcal{Q}(\mathcal{M}')$. For plan-result queries, consider that $\mathcal{Q}_{PR}(\mathcal{M}^{\mathcal{A}}) = \langle \ell, \langle p_1, \ldots, p_k \rangle \rangle$ and $\mathcal{Q}_{PR}(\mathcal{M}') = \langle \ell', \langle p'_1, \ldots, p'_j \rangle \rangle$. Now we can say that $\mathcal{Q}_{PR}(\mathcal{M}^{\mathcal{A}}) \models \mathcal{Q}_{PR}(\mathcal{M}')$ iff $(\ell = \ell')$ and $\wedge_i p_i = \wedge_i p'_i$. For the plan-executability queries consider that $\mathcal{Q}_{PE}(\mathcal{M}^{\mathcal{A}}) = \langle \ell, a, p \rangle$ and $\mathcal{Q}_{PE}(\mathcal{M}') = \langle \ell', a', q \rangle$. Now we can say that $\mathcal{Q}_{PE}(\mathcal{M}^{\mathcal{A}}) \models \mathcal{Q}_{PE}(\mathcal{M}')$ iff $(\ell = \ell')$ and $a = a'$ and $p \models q$.

Definition 8. *Given an agent-interrogation task $\langle \mathcal{M}^{\mathcal{A}}, \mathbb{Q} \rangle$, two models M_i and M_j are **prunable** denoted as $\mathcal{M}_i \langle \rangle \mathcal{M}_j$, iff $\exists Q \in \mathbb{Q} : M_i \rfloor^Q M_j$ and $(Q(\mathcal{M}^{\mathcal{A}}) \models Q(M_i)$ and $Q(\mathcal{M}^{\mathcal{A}}) \not\models Q(M_j))$ or $(Q(\mathcal{M}^{\mathcal{A}}) \not\models Q(M_i)$ and $Q(\mathcal{M}^{\mathcal{A}}) \models Q(M_j))$.*

3.3 Solving the Interrogation Task

Algorithm 1 shows our overall algorithm for interrogating autonomous agents. It takes the agent $\mathcal{A}$, the set of propositions $\mathbb{P}$, and set of all actions $\mathbb{A}$ as input and gives the set of estimated models represented by Λ_{est} as output. We initialize Λ_{est} as empty set (line 1) representing that we are starting at the most abstract node in model lattice.

In each iteration of the main loop (line 2), we keep track of the current node in the lattice. We pick a pal tuple γ corresponding to one of the descending edges in the lattice from n given by some input ordering of Γ. The correctness of the algorithm does not depend on this ordering. We then generate all the sets of λ tuples at the current node represented by the label ℓ_N of the node (line 3). The inner loop (line 4) iterates over the set of all possible models represented by set of estimated tuples Λ_{est}. Each abstract model represented by Λ is then refined with the pal tuple γ giving three different models and form pairs from these models and iterate over these pairs (line 5). Here $\mathcal{M}^+_\gamma$, $\mathcal{M}^-_\gamma$, and $\mathcal{M}^\phi_\gamma$ represents the abstract models equivalent to $\Lambda \cup \{\gamma^+\}$, $\Lambda \cup \{\gamma^-\}$, and $\Lambda \cup \{\gamma^\phi\}$ respectively.

For each pair, we generate a query $\mathcal{Q}$ using *generate_query()* that can distinguish between the models in that pair. We then call *filter_models()* which poses the query $\mathcal{Q}$ to the agent and the two models. Based on their responses, we prune the models whose responses were not consistent with that of the agent (line 8). Then we update the estimated set of models represented by Λ_{est} at the level of abstraction of node n. We continue this process until we reach the most concretized node of the lattice (meaning all possible model components $\lambda \in \Lambda$ are refined). The remaining set of models represents the estimated set of models for the agent. This algorithm would require $O(|\mathbb{A}| \times |\mathbb{P}|)$ queries. However, our empirical studies show that we never generate so many queries.

Section 3.4 describes the *generate_query()* (line 6) component of the algorithm, and Section 3.5 descibes the *filter_models()* (line 7) component.

Algorithm 1: Agent Interrogation Algorithm

Input: $\mathcal{A}, \mathbb{A}, \mathbb{P}$
Output: Λ_{est}

1 $\Lambda_{est} = \{\{\phi\}\}$;
2 **for** γ *in some input ordering of* Γ **do**
3 $\quad \ell_N \leftarrow \Lambda_{est} \times \{\gamma^+, \gamma^-, \gamma^\phi\}$;
4 $\quad$ **for** *each* Λ *in* Λ_{est} **do**
5 $\quad\quad$ **for** *each pair* $\{\mathcal{M}_i, \mathcal{M}_j\}$ *in* $\{\mathcal{M}^+_\gamma, \mathcal{M}^-_\gamma, \mathcal{M}^\phi_\gamma\}$ **do**
6 $\quad\quad\quad \mathcal{Q} \leftarrow$ generate_query$(\mathcal{M}_i, \mathcal{M}_j)$;
7 $\quad\quad\quad \ell_N^{prune} \leftarrow$filter_models$(\mathcal{Q}, \mathcal{M}^{\mathcal{A}}, \mathcal{M}_i, \mathcal{M}_j)$;
8 $\quad\quad\quad \ell_N \leftarrow \ell_N \setminus \ell_N^{prune}$
9 $\quad\quad$ **end**
10 $\quad$ **end**
11 $\quad \Lambda_{est} \leftarrow \ell_N$;
12 **end**

3.4 Query Generation

The query generation process corresponds to *generate_query()* module in algorithm 1 which takes two models $\mathcal{M}_i$ and $\mathcal{M}_j$ as input and generates a query $\mathcal{Q}$ that can distinguish between them, and if possible, satisfy prunability condition too. As we have two classes of possible queries, we describe a separate procedure to generate queries of each type.

Plan Result Queries Intuitively plan-result queries distinguish between the models based on action effects in both of them.

We can reduce the plan-result query generation to a planning problem. The idea is to maintain a separate copy $\mathcal{P}^{\mathcal{M}_i}$ and $\mathcal{P}^{\mathcal{M}_j}$ of all the propositions $\mathcal{P}$, and formulate each precondition and effect of an action as a conjunction of predicates in both the copies of the propositions.

Let the planning problem $P_{PR} = \langle \mathcal{M}^{PR}, s_{\mathcal{I}}, s_{\mathcal{G}} \rangle$, where $\mathcal{M}^{PR}$ is a model with propositions $\mathcal{P}^{PR} = \mathcal{P}^{\mathcal{M}_i} \cup \mathcal{P}^{\mathcal{M}_j}$, and actions $\mathbb{A}$ where for each action $a \in \mathbb{A}$, *pre*(a) = *pre*$(a^{\mathcal{M}_i}) \wedge$ *pre*$(a^{\mathcal{M}_j})$ and *eff*(a) = *eff*$(a^{\mathcal{M}_i}) \wedge$ *eff*$(a^{\mathcal{M}_j})$. $s_{\mathcal{I}} = s_{\mathcal{I}}^{\mathcal{M}_i} \wedge s_{\mathcal{I}}^{\mathcal{M}_j}$ is the initial state where $s_{\mathcal{I}}^{\mathcal{M}_i}$ and $s_{\mathcal{I}}^{\mathcal{M}_j}$ are different copies of all predicates in the initial state, and $s_{\mathcal{G}}$ is the goal state and it is expressed as $\exists p \ (p^{\mathcal{M}_i} \wedge \neg p^{\mathcal{M}_j}) \vee (\neg p^{\mathcal{M}_i} \wedge p^{\mathcal{M}_j})$.

With this formulation whenever we have at least one action in both the models which has different effects in both of them, the goal will be reached. For example, consider the models $\mathcal{M}^{\mathcal{A}}$ and $\mathcal{M}_1$ mentioned in figure 2. On applying the $pick\text{-}b1$ action from the state where the action can be applied in both the models, one of them will lead to $in\text{-}hand\text{-}b1$ being true and the other will not. Hence starting with an initial state $s_{\mathcal{I}} = on\text{-}floor\text{-}b1 \wedge handempty$, the plan to reach the goal will be $pick\text{-}b1$. The following theorem formalizes this idea.

Theorem 9. *Given a pair of models $\mathcal{M}_i$ and $\mathcal{M}_j$, the planning problem P_{PR} has a solution iff $\mathcal{M}_i$ and $\mathcal{M}_j$ have a distinguishing plan-result query $\mathcal{Q}_R$.*

Proof (Sketch). We use the fact that $\mathcal{Q}_R : \mathcal{M} \mapsto \{\ell, s_\mathcal{F}\}$. Here $\mathcal{Q}_R$ comprises of an initial state $s_\mathcal{I}$ and plan π. The initial state $s_\mathcal{I}$ in $\mathcal{Q}_R$ and P_{PR} is same. Starting with this initial state, an action becomes a part of the plan π only when it can be applied to both the models $\mathcal{M}_i$ and $\mathcal{M}_j$ independently as precondition of each action is conjunction of preconditions of the same action in $\mathcal{M}_i$ and $\mathcal{M}_j$ independently. Subsequently the goal state $s_\mathcal{G}$ of the planning problem is achieved when the effects of an action in the plan does not match. Hence if the planning problem P_{PR} gives a solution plan π, then there exists a query $\mathcal{Q}_R$. Also, as described in Section 3.2 whenever there exists a distinguishing plan-result query, the starting state $s_\mathcal{I}$ is part of $\mathcal{Q}_R$, and the way we generate the P_{PR} problem ensures we will get a plan π as the solution. □

Plan Executability Queries As opposed to plan-result queries, plan-executability queries distinguish between models differing in preconditions of some action, which affect the executability of plans given by the query.

We can reduce plan-executability query generation to a planning problem. Similar to plan-result queries, a separate copy of all the propositions is maintained.

Let the planning problem $P_{PE} = \langle \mathcal{M}^{PE}, s_\mathcal{I}, s_\mathcal{G} \rangle$, where $\mathcal{M}^{PE}$ is a model with propositions $\mathcal{P}^{PE} = \mathcal{P}^{\mathcal{M}_i} \wedge \mathcal{P}^{\mathcal{M}_j} \wedge p_\gamma$, and actions $\mathbb{A}$ where for each action $a \in \mathbb{A}$, $pre(a) = pre(a^{\mathcal{M}_i}) \vee pre(a^{\mathcal{M}_j})$ and $eff(a) = (when(pre(a^{\mathcal{M}_i}) \wedge pre(a^{\mathcal{M}_j}))(eff(a^{\mathcal{M}_i}) \wedge eff(a^{\mathcal{M}_j}))\ (when((pre(a^{\mathcal{M}_i}) \wedge \neg pre(a^{\mathcal{M}_j})) \vee (\neg pre(a^{\mathcal{M}_i}) \wedge \neg pre(a^{\mathcal{M}_j})))\ (p_\gamma))$, $s_\mathcal{I} = s_\mathcal{I}^{\mathcal{M}_i} \wedge s_\mathcal{I}^{\mathcal{M}_j}$ is the initial state where $s_\mathcal{I}^{\mathcal{M}_i}$ and $s_\mathcal{I}^{\mathcal{M}_j}$ are different copies of all predicates in the initial state, and $s_\mathcal{G}$ is the goal state and it is expressed as p_γ.

With this formulation whenever we have an action a which cannot be applied in the same state s_d in both the models, the planner will generate a plan including from the initial state to state s_d, and append action a to it. This new plan will be the solution to the planning problem P_{PE}. For example, consider the models $\mathcal{M}_1$ and $\mathcal{M}_2$ mentioned in figure 2. In a state where $on\text{-}floor\text{-}b1$ and $handempty$ are true, we can apply $pick\text{-}b1$ in $\mathcal{M}_1$ but not in $\mathcal{M}_2$. Hence for an initial state $s_\mathcal{I} = on\text{-}floor\text{-}b1 \wedge handempty$, the plan to reach the goal will be $pick\text{-}b1$. The following theorem formalizes this notion.

Theorem 10. *Given a pair of models $\mathcal{M}_i$ and $\mathcal{M}_j$, the planning problem P_{PE} has a solution iff $\mathcal{M}_i$ and $\mathcal{M}_j$ have a distinguishing plan-executability query $\mathcal{Q}_E$.*

Proof (Sketch). $\mathcal{Q}_E$ comprises of an initial state $s_\mathcal{I}$ and plan π. The initial state $s_\mathcal{I}$ in $\mathcal{Q}_E$ and P_{PE} is same. Starting with this initial state, an action becomes a part of the plan π only when it can be applied any one or both of the models $\mathcal{M}_i$ and $\mathcal{M}_j$. So two cases arise here, if the action can be executed in both the models, the effect of both the actions is applied to the state and next action is searched. Otherwise if the action is applicable only in one of the models, but not the other, the effect of the action is a dummy proposition p_γ which is also the goal. So as soon an action is found that is possible in one of the models but not the other, the resulting plan becomes the plan needed by query $\mathcal{Q}_E$. Hence if the planning problem P_{PE} gives a solution plan π, then there exists a query $\mathcal{Q}_E$ that consists of $s_\mathcal{I}$ and π as input.

Also, as described in Section 3.2 of the paper, whenever there exists a distinguishing plan-executability query, the starting state $s_\mathcal{I}$ is part of $\mathcal{Q}_E$, and the way we generate the P_{PE} problem ensures we will get a plan π as the solution. □

3.5 Filtering Possible Models

This section describes the *filter_models()* module in algorithm 1 which takes as input the agent model $\mathcal{M}^\mathcal{A}$, the two models being compared $\mathcal{M}_i$ and $\mathcal{M}_j$, and the query $\mathcal{Q}$ (generated by the *generate_query()* module explained in the previous section), and gives the subset ℓ_N^{prune} which is not consistent with the agent model.

Firstly, the algorithm *asks the query* Q to both the models $\mathcal{M}_i$ and $\mathcal{M}_j$ and the agent $\mathcal{M}^\mathcal{A}$. Based on the responses of all three, it determines if the two models are prunable, i.e. $\mathcal{M}_i \langle\rangle \mathcal{M}_j$. As mentioned in Def. 8, checking for prunability involves checking if the responses to the query $\mathcal{Q}$ by one of the models $\mathcal{M}_i$ or $\mathcal{M}_j$ is consistent with that of the agent or not.

If the models are prunable, let the model not consistent with agent be $\mathcal{M}'$ where $\mathcal{M}' \in \{\mathcal{M}_i, \mathcal{M}_j\}$. Now recall that a model is a set of palm tuples. As shown in figure 3, based on response to a query, if a model is found to be inconsistent for the first time at a node n in the lattice, with an incoming edge of label γ, any model with same mode of γ as $\mathcal{M}'$ will also be inconsistent. This is because a plam tuple uniquely identifies the mode in which a predicate will appear in an action's location which can be precondition or effect. And since this tuple is inconsistent with the agent, any model containing this will also need the same mode of predicate in that action's precondition or effect. This idea paves way for the concept of partitions which is discussed below.

Given lattice nodes n_i and n_j, the edge $n_j \rightarrow n_i$ labeled γ, and the set Λ of palm tuples present at the parent node n_j, a **partition** of node n_i is the set of disjoint subsets $\Lambda \cup \{\gamma^+\}$, $\Lambda \cup \{\gamma^-\}$, and $\Lambda \cup \{\gamma^\phi\}$.

So depending on the model $\mathcal{M}'$ which is inconsistent with agent model $\mathcal{M}^\mathcal{A}$, we can prune out the whole partition containing $\mathcal{M}'$. This partition is returned by *filter_models()* module as ℓ_N^{prune}.

If the models are not prunable, i.e. the query is not executable on agent $\mathcal{A}$, we do not discard any partitions. But if no prunable query is possible, i.e. the palm tuple set Λ being considered is last in Λ_{est}, we use the response of the plan-executability query to prune the partitions. Recall that in response to the plan-executability query we get the failed action a_{Fail} and the literal $\{p_{Fail}, m_{Fail}\}$ which was present in that action's precondition which was not met. This gives us an opportunity to refine the models in terms of a new palm tuple $\langle p_{Fail}, a_{Fail}, l = precondition, m_{Fail} \rangle$. So even if we are unable to generate a plan that can be executed by the agent, we refine with respect to this new palm tuple.

3.6 Correctness of Agent Interrogation Algorithm

In this section we prove that the set of estimated models returned by the agent interrogation algorithm are correct. A good starting point will be to verify the correctness of the queries. In the following theorem we show that if we pick the models from different partitions to generate the query (Step 5 of the algorithm 1), then we will always get distinguishing queries. We break this into 2 parts, first we consider the case where the location l in the pal tuple is precondition, then we discuss about the pal tuples with effect as the location.

Theorem 11. *For the refinement in terms of pal tuple $\gamma = \langle p, a, l = precondition\rangle$ of two models $\mathcal{M}_i$ and $\mathcal{M}_j$, if $\mathcal{M}_i$ and $\mathcal{M}_j$ are not distinguishable $\mathcal{M}_i \not\chi \mathcal{M}_j$, then their refinements when adding γ will be distinguishable only if the refinements belong to different partitions i.e., $(\mathcal{M}_i \cup \gamma^{m_1}) \not\chi (\mathcal{M}_j \cup \gamma^{m_2})$ if $m_1 \neq m_2$ and $(\mathcal{M}_i \cup \gamma^{m_1}) \not\chi (\mathcal{M}_j \cup \gamma^{m_2})$ if $m_1 = m_2$, where $m_1, m_2 \in \{+, -, \phi\}$.*

Proof (Sketch). Let us consider 2 different cases;
First case when $m_1 = m_2$, i.e., both $(\mathcal{M}_i \cup \gamma^{m_1})$ and $(\mathcal{M}_j \cup \gamma^{m_2})$ are in the same partition. Let us assume that a query $\mathcal{Q}$ with plan $\pi_{\mathcal{Q}}$ exists that can distinguish between the refined models $(\mathcal{M}_i \cup \gamma^{m_1})$ and $(\mathcal{M}_j \cup \gamma^{m_2})$. Now since before refinement $\mathcal{M}_i \not\chi \mathcal{M}_j$ and the only change we have made in the models is in action a, it should be part of the plan π. More specifically it should be the last action in the plan as after applying this action, both models will result in different states. But the change being made in both $\mathcal{M}_i$ and $\mathcal{M}_j$ is same, hence this action a cannot distinguish between the two models, so we reach a contradiction with our initial assumption, hence $(\mathcal{M}_i \cup \gamma^{m_1}) \not\chi (\mathcal{M}_j \cup \gamma^{m_2})$ if $m_1 = m_2$.

For the second case when $m_1 \neq m_2$ i.e., $(\mathcal{M}_i \cup \gamma^{m_1})$ and $(\mathcal{M}_j \cup \gamma^{m_2})$ are in different partitions, we can safely assume that we have at least one action a which is part of γ that can distinguish between the two models. When $l = precondition$, we can generate a state $s_{\mathcal{I}}$ where this action is applicable in one model but not in another. More specifically, if $\{m_1, m_2\} = \{+, -\}$, we can generate an initial state $s_{\mathcal{I}}$ with literal p or $\neg p$, in both the cases, only one of the models will be able to execute the plan π. If $\{m_1, m_2\} = \{+, \phi\}$, we can generate an initial state $s_{\mathcal{I}}$ with literal $\neg p$, and hence only the model with $m = \phi$ will be able to execute the plan π to completion. And finally if $\{m_1, m_2\} = \{-, \phi\}$, we can generate an initial state $s_{\mathcal{I}}$ with literal p, and hence only the model with $m = \phi$ will be able to execute the plan π to completion. Note that for these cases the distinguishing query will be plan-executability query $\mathcal{Q}_E$. □

We cannot use the exact same argument when refinement is done for the effects because we can have models with different effects that are functionally equivalent. An intuitive example of this will be a pair of models where in one of the model the same literal appears in both the preconditions and effects, whereas in the other model the same literal appears only in the precondition.

For example, consider the chemistry lab robot that cross-checks for the labels of the bottles when placing the chemical bottles in correct shelves. Assume we have two models $\mathcal{M}_1$ and $\mathcal{M}_2$ with the two versions of the action *put-on-shelf-b1-s1* as shown in figure 4. In this case, we cannot generate an initial state $s_{\mathcal{I}}$ that can lead to plan that can distinguish between the models with these two different refinements. In simpler terms, the planning problem P_{PE} discussed in Theorem 10 will not give a solution in this case.

(a) $\mathcal{M}_1$'s *put-on-shelf-b1-s1* action
$\lambda^{\phi} = \langle$*same-label-b1-s1, put-on-shelf-b1-s1, eff,*$\phi\rangle$

(in-hand-b1),	→	(not(in-hand-b1)),
(same-label-b1-s1)		(handempty),
		(on-shelf-b1-s1)

(b) $\mathcal{M}_2$'s *put-on-shelf-b1-s1* action
$\lambda^{+} = \langle$*same-label-b1-s1, put-on-shelf-b1-s1, eff,*$+\rangle$

(in-hand-b1),	→	(not(in-hand-b1)),
(same-label-b1-s1)		(handempty),
		(on-shelf-b1-s1),
		(same-label-b1-s1)

Figure 4: Two model variants that are functionally equivalent

Theorem 12. *For the refinement in terms of pal tuple $\gamma = \langle p, a, l = \text{effect}, m\rangle$ of two models $\mathcal{M}_i$ and $\mathcal{M}_j$, if $\mathcal{M}_i$ and $\mathcal{M}_j$ are not distinguishable $\mathcal{M}_i \not\chi \mathcal{M}_j$, then their refinements when adding palm tuple λ will be distinguishable only if the refinements belong to separate partitions except when one of the partition corresponds to tuple γ^{ϕ} and $\gamma' = \langle p, a, l = precondition, m = \{+, -\}\rangle \in \mathcal{M}_i \cap \mathcal{M}_j$.*

Proof (Sketch). Let us consider 2 different cases; First case when $m_1 = m_2$, i.e., both $(\mathcal{M}_i \cup \lambda^{m_1})$ and $(\mathcal{M}_j \cup \lambda^{m_2})$ are in the same partition. Let us assume that a query $\mathcal{Q}$ with plan $\pi_{\mathcal{Q}}$ exists that can distinguish between the refined models $(\mathcal{M}_i \cup \gamma^{m_1})$ and $(\mathcal{M}_j \cup \gamma^{m_2})$. Now since before refinement $\mathcal{M}_i \not\chi \mathcal{M}_j$ and the only change we have made in the models is in action a, it should be part of the plan π. More specifically it should be the last action in the plan as this action cannot be applied in one of the models. But the change being made in both $\mathcal{M}_i$ and $\mathcal{M}_j$ is same, hence this action a cannot distinguish between the two models, so we reach a contradiction with our initial assumption, hence $(\mathcal{M}_i \cup \gamma^{m_1}) \not\chi (\mathcal{M}_j \cup \gamma^{m_2})$ if $m_1 = m_2$.

For the second case when $m_1 \neq m_2$ i.e., $(\mathcal{M}_i \cup \gamma^{m_1})$ and $(\mathcal{M}_j \cup \gamma^{m_2})$ are in different partitions, if we consider only the cases where the following condition holds for two palm tuple variants being considered: $\{\langle p, a, l = precondition, m_1\rangle \in \mathcal{M}_i \cap \mathcal{M}_j\} \wedge \{\langle p, a, l = \textit{effect}, m_2\rangle \in \mathcal{M}_i \cap \mathcal{M}_j\} \Rightarrow m_1 \neq m_2$; then we can safely assume that we have at least one action a which is part of γ that can distinguish between the two models. When $l =$ *effect*, we can generate a state $s_{\mathcal{I}}$ where this action is applicable, then according to Theorem 9, we can get always get a plan π that will distinguish between the two models. Note here that for these cases the distinguishing query will be plan-result query $\mathcal{Q}_R$. □

The theorems given above prove that the way we pick models to generate a query gives us distinguishing queries

in almost all cases. And for the cases it does not generate a distinguishing query, we do not prune any model. We now prove that the algorithm prunes away a model only when it is inconsistent with the responses given by the agent. Here we are assuming that the agent has deterministic actions, so we can safely infer that an inconsistent answer will always be inconsistent.

Theorem 13. *If we prune away an abstract model $\mathcal{M}^{abs}$, then no possible concretization of $\mathcal{M}^{abs}$ will result into a model consistent with the agent model $\mathcal{M}^{\mathcal{A}}$.*

Proof (Sketch). At each node in the lattice, we always prune away some of the models, this pruning happens in one of the two possible ways:

First, when the distinguishing query between two models (which are variants of each other) is executed successfully by the agent $\mathcal{A}$, we discard a model that is not consistent with the agent. We discard a model only when there is a proposition in the abstracted version of the final state of the agent that is not present in the final state of the model. So for any concretized version of the model, the number of states reachable will never increase, so any concretized version of that model cannot make those propositions (which were false when we initially discarded) true, so it is never possible for a concretized version of a discarded model to be always consistent with the agent model.

Second case is easier to deal here. Whenever we are not able to generate a plan that is fully executable on the agent $\mathcal{A}$, we get a new refinement tuple $\langle p_{Fail}, a_{Fail}, l = precondition, m_{Fail}\rangle$. This is akin to the agent telling us directly that this is the correct form of this palm tuple, hence we can discard other variants of this tuple with full surety. □

With the guarantee that we are not pruning away any correct possible model, we now move on to prove that the agent interrogation algorithm will terminate, hence giving a solution.

Theorem 14. *The Agent Interrogation Algorithm mentioned in algorithm 1 will always terminate.*

Proof (Sketch). At each step of the algorithm, when we consider a refinement in terms of γ tuples, we are left with one or more variant of the γ tuple. This ensures that we never have to refine models more than once at a single level in the lattice. We are assuming level in our subset lattice, to be the number of refined γ tuples. Since we refine at least one γ tuple in every iteration of the algorithm, the algorithm is bound to terminate as the number of γ tuples is finite for a finite number of propositions and actions under consideration. □

In the last theorem we proved that agent interrogation algorithm will always give us a solution, we now prove that the solution given by the algorithm is correct.

Theorem 15. *As part of its solution, Agent Interrogation algorithm always gives a set of models that contains the agent's true model $\mathcal{M}^{\mathcal{A}}$.*

Domain	$\lvert\mathbb{P}^*\rvert$	$\lvert\mathbb{A}\rvert$	$\lvert\overline{\mathcal{M}}\rvert$	$\lvert\overline{\mathcal{Q}}\rvert$	Algorithm 1	
					$\lvert\hat{\mathcal{Q}}\rvert$	Time/$\mathcal{Q}$ (sec)
gripper	5	3	9^{15}	$15 * 2^5$	34	0.14
blocks*	9	4	9^{36}	$36 * 2^9$	85	2.56
elevator	10	4	9^{40}	$40 * 2^{10}$	78	6.10
logistics	11	6	9^{66}	$66 * 2^{11}$	48	10.99
parking	18	4	9^{72}	$72 * 2^{18}$	144	8.13
satellite	17	5	9^{85}	$85 * 2^{17}$	72	14.35
stacks$	10	12	9^{120}	$120 * 2^{10}$	122	10.69

Table 1: Table showing the number of queries $\lvert\hat{\mathcal{Q}}\rvert$ asked to the agent in our approach, as opposed to $\lvert\overline{\mathcal{Q}}\rvert$ asked in the brute force solution having $\lvert\overline{M}\rvert$ possible models to start with. Time shown is the average time taken per query for 5 runs of the agent interrogation algorithm.
Full name of IPC domain is *blocks-world, $openstacks.

Proof (Sketch). As a solution to the agent interrogation algorithm, we might get multiple possible models each of which are functionally equivalent. Theorem 14 sets up an invariant for the agent interrogation algorithm that is the number of refined palm tuples increase as the number of iterations of the algorithm increase. This property when combined with theorem 13 ensures that at each step only the incorrect models are discarded. So the models leftover at the most concretized node after all the palm tuples are refined are going to be the set of models which the algorithm could not discard, hence one of these models is guaranteed to be the agent model. □

4 Empirical Evaluation

Since the problem we address has not been addressed in prior work, no benchmarks or baselines are available.

A naive way to solve the problem is to have a brute force solution where all the possible models can be generated and then their answers to queries are compared to agent's answers. This method is guaranteed to find the solution but the complexity of this approach is exponential in terms of number of predicates. As pointed out earlier, the number of possible models is $9^{|\mathbb{A}|\times|\mathbb{P}|}$.

In order to apply our framework to the relational models, we use the following method. Consider the case where empty bottles in the lab can be stacked on top of each other. Consider the action $stack$ which takes two parameters $?b1$ and $?b2$. Assume we use the predicate $on(?x, ?y)$ to represent bottle $?x$ is on top of bottle $?y$. Here on can have 2 variations, $on(?b1, ?b2)$ and $on(?b2, ?b1)$. Hence we consider these two as different lifted predicates. The number of such predicates is denoted as $|\mathbb{P}^*|$.

Note that the number of queries is not affected by the number of objects as the approach finds the minimum number of objects needed to distinguish between the abstracted models and uses it, and providing more objects does not change the behaviour of the algorithm.

To see how the method scales with an increase in number of predicates and actions, we tested the implementation with

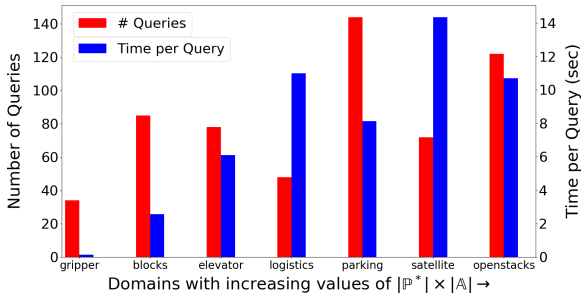

Figure 5: Bar chart showing the number of queries and time per query for the seven IPC domains. The domains are arranged in increasing order of $|\mathbb{P}^*| \times |\mathbb{A}|$ from left to right

7 IPC domains as the agent models. Table 1 shows the results for various domains. Note that the number of queries asked $|\hat{\mathcal{Q}}|$ are smaller than the upper bound $4 \times |\mathbb{P}^*| \times |\mathbb{A}|$ possible for each of the case. We get this upper bound as each *pal* tuple will need at max 2 queries to distinguish between 3 of its variations. Also, the number of expected models was always one as the corner case shown in figure 4 was not present in any of the domains. Also, there is no definite pattern in the number of queries asked as the order in which queries were asked (depending of ordering of γ tuples) was random. So a better query asked earlier in the interrogation process can lead to small number of queries asked.

5 Conclusion

We have presented a novel approach for estimating the model of an autonomous agent by interrogating it. In this paper we showed that the number of queries required to estimate the model is dependent only on the number of actions and predicates, and is independent of the size of the environment. Also, this approach requires much smaller number of queries as compared to data required by other approaches.

Extending this approach to more general types of agents and environments featuring partial observability and/or non-determinism is a promising direction for future work.

Although our interface is a set of plans represented as logical statements, other works have explored using natural language as a way to provide plans as input (Lindsay et al. 2017). In future work, this can be used to extend our work thereby making the communication more realistic and close to how a human might actually interact with an autonomous agent.

References

Aineto, D.; Jiménez, S.; Onaindia, E.; and Ramírez, M. 2019. Model recognition as planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, 13–21.

Aineto, D.; Jiménez, S.; and Onaindia, E. 2018. Learning strips action models with classical planning. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*.

Alterman, R. 1986. An adaptive planner. In *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, AAAI'86, 65–69. AAAI Press.

Bäckström, C., and Jonsson, P. 2013. Bridging the gap between refinement and heuristics in abstraction. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, 2261–2267. AAAI Press.

Cresswell, S., and Gregory, P. 2011. Generalised domain model acquisition from action traces. In *International Conference on Automated Planning and Scheduling*.

Cresswell, S.; McCluskey, T.; and West, M. 2009. Acquisition of object-centred domain models from planning examples. In *International Conference on Automated Planning and Scheduling*.

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.

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. 2017. Flexible abstraction heuristics for optimal sequential planning. In *International Conference on Automated Planning and Scheduling*.

Khardon, R., and Roth, D. 1996. Reasoning with models. *Artif. Intell.* 87(1-2):187–213.

Lindsay, A.; Read, J.; Ferreira, J.; Hayton, T.; Porteous, J.; and Gregory, P. 2017. Framer: Planning models from natural language action descriptions. In *International Conference on Automated Planning and Scheduling*.

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 *AAAI Conference on Artificial Intelligence*.

Stern, R., and Juba, B. 2017. Efficient, safe, and probably approximately complete learning of action models. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 4405–4411.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted max-sat. *Artif. Intell.* 171(2-3):107–143.