Learning Generalized Models
by Interrogating Black-Box Autonomous Agents

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

Abstract
This paper develops a new approach for estimating the in-
ternal model of an autonomous agent that can plan and act,
by interrogating it. In this approach, the user may ask an au-
tonomous agent a series of questions, which the agent an-
wers 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 incon-
sistent 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 per-
vasive problem: how would we ascertain whether an au-
tonomous 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 de-
signed by their users; their internal software may be unavail-
able 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 au-
tonomous agent, essentially by interrogating it. The result-
ning model could be used to solve and predict how the agent
would act in a range of planning problems. Consider a sit-
uation 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 fol-
lowed by bottle 2 and bottle 3 without putting down any-
thing in between?” Answers to such questions help the re-
searcher develop an understanding of the assistant’s frame
of knowledge, or “model”, while placing a minimal intro-
spective requirement on the agent. Most simulator-based and
analytical-model based agents can easily answer such ques-
tions. 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 di-
rection of research is on solving this problem.

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 con-
junctive preconditions, add lists, and delete lists (and that
the agent’s model is expressible as such), although our frame-
work can be extended to handle other types of formal do-
mam 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 ac-
tion 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 ap-
proaches. E.g., three actions and five atomic propositions
yield $10^{14}$ possible models. We make minimal assump-
tions about the agent’s question-answering capabilities: we
assume that it can answer queries about hypothetical execu-
tions, 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 organized as follows. The next 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.

2 Related Work

Our solution approach takes motivation from action model learning, which has been used extensively to learn domain models by analyzing the actions of the agent (Alterman 1986; Yang, Wu, and Jiang 2007; Stern and Juba 2017; Aineto, Jimenez, and Onaindia 2018).

Simultaneous Learning and Filtering (SLAF) (Amir and Chang 2008) is another approach which uses logical filtering (Amir and Russell 2003) to learn partially observable action models from action and observation traces.

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.

Camacho et al. (Camacho and McIlraith 2019) present a new approach for estimating LTL representations of planning models. However, their approach requires an oracle with knowledge of the LTL form of the model for the equivalence queries. In contrast, our approach does not require the agent to know the representation that the user wishes to use for estimating its model.

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. However, these methods 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 A. A human interrogator H’s task is to compute a planning model M_A that captures A’s internal model. The only information H has is the set of actions A that A can perform. Since A has functional definitions of the predicates in H’s vocabulary, i.e. A and H agree on the propositions P, there is sufficient information for a dialog between H and 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 \(|P| \times |A|\). This is because using one proposition and one action, we can generate 9 different models as each proposition can appear in 3 different ways (positive, negative, or absent) in an action in two places, precondition, and effect.

Our approach iteratively generate pairs of abstract models and eliminates one of them by asking 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 M_1 and M_2 having only the propositions handempty, on-floor-b1 and on-floor-b2 and the agent’s model is M_A (Fig 2). H can ask the agent

| Query | Model M_A | Model M_1 | Model M_2 |
|-------|------------|------------|------------|
| (handempty), (on-floor-b1) → (in-hand-b1), ¬(handempty), ¬(on-floor-b1) | (handempty), (on-floor-b1) → (in-hand-b1), (handempty), (on-floor-b1) | (handempty), (on-floor-b1) → (in-hand-b1), ¬(handempty), (on-floor-b1) |
| [handempty], (on-floor-b2) → (in-hand-b2), ¬(handempty), ¬(on-floor-b2) | (handempty), (on-floor-b2) → (in-hand-b2), (handempty), (on-floor-b2) | (handempty), (on-floor-b2) → (in-hand-b2), ¬(handempty), (on-floor-b2) |

Figure 2: pick actions of the agent model M_A and two abstract models M_1 and M_2. Here X → Y means that X is the precondition of an action and Y is the effect.
what will happen if 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 \( M_2 \), 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 \( A \)'s model using a STRIPS like planning model (Fikes and Nilsson 1971) represented as a pair \( \mathcal{M} = (\mathbb{P}, \mathbb{A}) \), where \( \mathbb{P} = \{p_1, \ldots, p_n\} \) is a finite set of propositions, each with a binary domain \( \text{dom}(p_i) \) associated with it; \( \mathbb{A} = \{a_1, \ldots, a_k\} \) is a finite set of actions, each represented as a tuple \( \langle \text{pre}(a_j), \text{eff}(a_j) \rangle \). For each action \( a_j, \text{pre}(a_j) \) and \( \text{eff}(a_j) \) represent conjunctive sets of literals. In this representation, states are represented as (conjunctive) sets \( s \) of true propositions. Let \( \text{eff}^+(a) \) (\( \text{eff}^-(a) \)) denote the positive (negative) literals in \( \text{eff}(a) \). The effect of an action \( a \) on a state \( s \in 2^p \) is represented as \( a(s) = (s \cup \text{eff}^+(a)) \setminus \text{eff}^-(a) \) if \( \text{pre}(a) \subseteq s \); \( a(s) \) is undefined otherwise. A planning problem is defined as a 3-tuple \( (\mathcal{M}, s_I, s_G) \) where \( \mathcal{M} = (\mathbb{P}, \mathbb{A}) \) is the planning model, \( s_I \in 2^p \) is the initial state, \( s_G \subseteq 2^p \) is the goal condition. A plan \( \pi \) of length \( k \) is a sequence of actions \( \langle a_1, \ldots, a_k \rangle \) that solves this planning problem if \( s_G \subseteq \pi(s_I) \). Here \( \pi(s) = a_k \circ \ldots \circ a_1(s) \). 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 \( (\mathcal{M}^A, \mathcal{Q}) \) where \( \mathcal{M}^A \) is the model of the agent (unknown to the interrogator) being interrogated, and \( \mathcal{Q} \) is the class of queries that can be posed to the agent by the interrogator.

Let \( \Theta \) 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 \( \theta^* : \mathcal{Q} \cup \{\text{Stop}\} \rightarrow \Theta^* \) 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 \( p \in \mathbb{P}, a \in \mathbb{A}, l \in \{\text{pre}, \text{eff}\} \), 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 \( \lambda \) in a model denotes the fact that in that model, the proposition \( \lambda_p \) appears in an action \( \lambda_a \) at a location \( \lambda_l \) as a true (false) literal when sign \( \lambda_m \) is positive (negative), and is absent when \( \lambda_m = \phi \). This allows us to define the set-minus operation \( \mathcal{M} \setminus \lambda \) on this model as removing the palm-tuple \( \lambda \) 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., $\lambda_1 \sim \lambda_2 \iff (\lambda_{1p} = \lambda_{2p}) \land (\lambda_1 = \lambda_2) \land (\lambda_{1n} \neq \lambda_{2n})$.

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; Helmer 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 $\lambda$ tuple to create an abstraction.

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

We also use the notation $M' \sqsubseteq \lambda M$ to represent the situation where $f_\lambda(M) = M'$.

We use this abstraction framework to define a subset-lattice over abstract models (Fig. 4a). Note that at each node we can have all possible variants of a palm tuple. For example the topmost node in Fig. 4b, we can have models corresponding to $\gamma^+_1$, $\gamma^-_1$, and $\gamma^\phi_1$. 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 $L$ is a 5-tuple $L = \langle N, E, \Gamma, \ell_N, \ell_E \rangle$, where $N$ is a set of lattice nodes, $\Gamma$ is the set of all pal tuples $\langle p, a, l \rangle$, $\ell_N : N \rightarrow 2^{2^3}$ is a node label function where $\Lambda = \Gamma \times \{+, -, \phi\}$, $E$ is the set of lattice edges, and $\ell_E : E \rightarrow \Gamma$ is a function mapping edges to edge labels such that for each edge $n_i \rightarrow n_j$, $\ell_n(n_i,j) = \{\Lambda \cup \{\gamma^\phi\}\}\{E \in \ell_N(n_i,j), \gamma = \ell(n_i \rightarrow n_j), k \in \{+, -, \phi\}\}$. The supremum $\top$ of the lattice $L$ 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 $L$ can be uniquely identified as the sequence of pal tuples that label edges leading to it from the supremum. As shown in Fig. 4b, even though theoretically $\ell : n \rightarrow 2^{2^3}$, only one of the sets is stored at each node as the others are pruned out based on $Q$. 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 $U$ be the set of possible models and $R$ a set of possible responses. A query $Q$ is a function $Q : U \rightarrow R$.

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 $\pi$.

Plan result queries $(Q_R)$ ask the agent for the state it will end up in if it executes the plan $\pi$ when starting in the state $s_I$. 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 $R_R$ for these queries is a tuple $(\ell, s_F)$, where $\ell$ is the number of steps for which the plan $\pi$ was successfully able to run, and $s_F \in 2^F$ is the final state of the agent after executing the query. If the plan $\pi$ is not executable according to the agent model $M^A$ then $\ell < \text{len}(\pi)$, otherwise if $\pi$ is executable then $\ell = \text{len}(\pi)$ and $s_F \in 2^F$ such that $M^A \models \pi(s_F) = s_F$. Thus, $Q_R : U \rightarrow N \times 2^F$, where $N$ is set of natural numbers.

Plan executability queries $(Q_E)$ ask the agent the length of the longest prefix of the plan $\pi$ that can be executed successfully when starting in the state $s_I$. 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 $lF$?” 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 $R_E$ for these queries is a tuple $(\ell, a_{\text{fail}}, p_{\text{fail}})$ where $\ell$ is the number of steps for which the plan $\pi$ was successfully able to run, $a_{\text{fail}}$ is the first action that failed to execute, and $p_{\text{fail}}$ is a literal in $a_{\text{fail}}$’s precondition that would not be satisfied at step $\ell < \text{len}(\pi)$. Essentially $a_{\text{fail}}$ and $p_{\text{fail}}$ are like error codes. Thus, $Q_E : M \rightarrow N \times X \times P$, where $N$ is 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 $Q$ is said to distinguish a pair of models $M_i$ and $M_j$, denoted as $M_i \not\models Q M_j$, iff $Q(M_i) \neq Q(M_j)$.

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 $M_i$ and $M_j$ are distinguishable, denoted as $M_i \not\models M_j$, iff there exists a query that can distinguish between them, i.e. $\exists Q(M_i) \not\models Q(M_j)$.
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 $M'$ is consistent with that of the agent, i.e. $Q(M^A) = Q(M')$. For plan-result queries, consider that $Q_{PR}(M^A) = (\ell, \{p_1, \ldots, p_k\})$ and $Q_{PR}(M') = (\ell', \{p'_1, \ldots, p'_l\})$. Now we can say that $Q_{PR}(M^A) = Q_{PR}(M')$ iff $(\ell = \ell')$ and $\Lambda p_i \equiv \Lambda p'_i$. For the plan-executability queries consider that $Q_{PE}(M^A) = (\ell, a, p)$ and $Q_{PE}(M') = (\ell', a', q)$. Now we can say that $Q_{PE}(M^A) = Q_{PE}(M')$ iff $(\ell = \ell')$ and $a = a'$ and $p \equiv q$.

**Definition 8.** Given an agent-interrogation task $(M^A, Q)$, two models $M_i$ and $M_j$ are prunable denoted as $M_i \parallel M_j$, iff $\exists Q \in \mathbb{Q} : M_i |^Q M_j$ and $(Q(M^A) = Q(M_i)$ and $Q(M^A) \neq Q(M_j))$ or $(Q(M^A) \neq Q(M_i)$ and $Q(M^A) = Q(M_j))$.

### 3.3 Solving the Interrogation Task

Algorithm 1 shows our overall algorithm for interrogating autonomous agents. It takes the agent $A$, the set of propositions $\mathbb{P}$, and set of all actions $\mathbb{A}$ as input and gives the set of estimated models represented by $\Lambda_{est}$ as output. We initialize $\Lambda_{est}$ 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 $\gamma$ corresponding to one of the descending edges in the lattice from $n$ given by some input ordering of $\Gamma$. The correctness of the algorithm does not depend on this ordering. We then generate all the sets of $\lambda$ tuples at the current node represented by the label $\ell_n$ of the node (line 3). The inner loop (line 4) iterates over the set of all possible models represented by set of estimated tuples $\Lambda_{est}$. Each abstract model represented by $\Lambda$ is then refined with the pal tuple $\gamma$ giving three different models and form pairs from these models and iterate over these pairs (line 5). Here $M^+_i$, $M^-_i$, and $M^0_i$ represents the abstract models equivalent to $\Lambda \cup \{\gamma^+\}$, $\Lambda \cup \{\gamma^-\}$, and $\Lambda \cup \{\gamma^0\}$ respectively.

For each pair, we generate a query $Q$ using `generate_query()` which can distinguish between the models in that pair. We then call `filter_models()` which poses the query $Q$ 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 $\Lambda_{est}$ 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 describes the `filter_models()` (line 7) component.

### 3.4 Query Generation

The query generation process corresponds to `generate_query()` module in algorithm 1 which takes two models $M_i$ and $M_j$ as input and generates a query $Q$ 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 $p^{M_i}$ and $p^{M_j}$ of all the propositions $\mathbb{P}$, 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 $P_{PR} = (M^{PR}, s_I, s_G)$, where $M^{PR}$ is a model with propositions $p^{PR} = p^{M_i} \cup p^{M_j}$, and actions $\lambda$ where for each action $a \in \lambda$, $pre(a) = pre(a^{M_i}) \land pre(a^{M_j})$ and $eff(a) = eff(a^{M_i}) \land eff(a^{M_j})$. $s_I = s_{I_i} \land s_{I_j}$ is the initial state where $s_{I_i}$ and $s_{I_j}$ are different copies of all predicates in the initial state, and $s_G$ is the goal state and it is expressed as $\exists p(p^{M_i} \land \neg p^{M_j}) \lor (\neg p^{M_i} \land p^{M_j})$.

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 $M^{A_i}$ and $M_j$ mentioned in figure 2. On applying the `pick-b1` action from the state where the action can be applied in both the models, one of them will lead to `in-hand-b1` being true and the other will not. Hence starting with an initial state $s_I = on-floor-b1 \land handempty$, the plan to reach the goal will be `pick-b1`. The following theorem formalizes this idea.

Proofs of all theorems are available in the full version of the paper at [https://bit.ly/2QFIPWD](https://bit.ly/2QFIPWD).

**Theorem 9.** Given a pair of models $M_i$ and $M_j$, the planning problem $P_{PR}$ has a solution iff $M_i$ and $M_j$ have a
distinguishing plan-result query \( Q_R \).

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 \( P_{PE} = (\mathcal{M}^{PE}, s_I^M, s_G^M) \), where \( \mathcal{M}^{PE} \) is a model with propositions \( p^{PE} = \mathcal{P}^{M_1} \land \mathcal{P}^{M_2} \land p_\gamma \), and actions \( \mathcal{A} \) where for each action \( a \in \mathcal{A} \), \( \text{pre}(a) = \text{pre}(a^{M_1}) \lor \text{pre}(a^{M_2}) \) and \( \text{eff}(a) = (\text{when}(\text{pre}(a^{M_1}) \land \text{pre}(a^{M_2}))(\text{eff}(a^{M_1}) \land \text{eff}(a^{M_2}))) \). Where \( (\text{when}(\text{pre}(a^{M_1}) \land \text{pre}(a^{M_2}))(\text{eff}(a^{M_1}) \land \text{eff}(a^{M_2}))) \) is the initial state where \( s_I^M = s_I^{M_1} \land s_I^{M_2} \) is the initial state, and \( s_G^M \) is the goal state and it is expressed as \( \rho_i \).

With this formulation whenever we have an action \( a \) which cannot be applied in the same state \( s_G^M \) in both the models, the planner will generate a plan including from the initial state to state \( s_G^M \) and append action \( a \) to it. This new plan will be the solution to the planning problem \( P_{PE} \). For example, consider the models \( M_1 \) and \( M_2 \) as mentioned in figure 2. In a state where \( \text{on-floor-b1} \) and \( \text{handempty} \) are true, we can apply \( \text{pick-b1} \) in \( M_1 \) but not in \( M_2 \). Hence for an initial state \( s_G^M = \text{on-floor-b1} \land \text{handempty} \), the plan to reach the goal will be \( \text{pick-b1} \). The following theorem formalizes this notion.

**Theorem 10.** Given a pair of models \( M_i \) and \( M_j \), the planning problem \( P_{PE} \) has a solution iff \( M_i \) and \( M_j \) have a distinguishing plan-executability query \( Q_{PE} \).

### 3.5 Filtering Possible Models

This section describes the filter\_models() module in algorithm 11 which takes as input the agent model \( \mathcal{M}^A \), the two models being compared \( M_i \) and \( M_j \), and the query \( Q \) (generated by the generate\_query() module explained in the previous section), and gives the subset \( \mathcal{P}^{\mathcal{M}^A} \) which is not consistent with the agent model.

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

If the models are pruneable, the model not consistent with agent be \( \mathcal{M}' \) where \( \mathcal{M}' \in \{ M_i, M_j \} \). 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 \( \gamma \), any model with same mode of \( \gamma \) as \( \mathcal{M}' \) will also be inconsistent. This is because a palm 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 \( n_j \to n_i \) labeled \( \gamma \), and the set \( \Lambda \) 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 \( \mathcal{M}^A \), we can prune out the whole partition containing \( \mathcal{M}' \). This partition is returned by filter\_models() module as \( \mathcal{P}^{\mathcal{M}^A} \).

If the models are not pruneable, i.e. the query is not executable on agent \( \mathcal{A} \), we do not discard any partitions. But if no pruneable query is possible, i.e. the palm tuple set \( \Lambda \) being considered is last in \( \Lambda_{\text{all}} \), 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 \( a_{\text{fail}} \) and the literal \( \{ p_{\text{fail}}, m_{\text{fail}} \} \) 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 \( \{ p_{\text{fail}}, a_{\text{fail}}, l = \text{precondition}, m_{\text{fail}} \} \). 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 11), 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 \( \gamma = \{ p, a, l = \text{precondition} \} \) of two models \( M_i \) and \( M_j \), if \( M_i \) and \( M_j \) are not distinguishable \( M_i \& M_j \), then their refinements when adding \( \gamma \) will be distinguishable only if the refinements belong to different partitions i.e., \( (M_i \cup \gamma_1) \not\subset (M_j \cup \gamma_2) \) if \( m_1 \neq m_2 \) and \( (M_i \cup \gamma_1) \not\subset (M_j \cup \gamma_2) \) if \( m_1 \neq m_2 \).

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 \( M_1 \) and \( M_2 \) 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 \( s_G \) that can lead to a 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.
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 \( \mathcal{M}_i \mathcal{M}_j \), then their refinements when adding palm tuple \( \lambda \) will be distinguishable only if the refinements belong to separate partitions except when one of the partition corresponds to tuple \( \gamma^0 \) and \( \gamma' = \langle p, a, l = \text{precondition}, m = \{+, -\} \in \mathcal{M}_i \cap \mathcal{M}_j \). 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 \( \mathcal{M}_A \).

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.

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 \( \mathcal{M}_A \).

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 \( 2^{|P|} \times |A| \).

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 \( \text{stack} \) which takes two parameters \(?b1\) and \(?b2\). Assume we use the predicate \( \text{on}(?,?) \) to represent bottle \(?x\) is on top of bottle \(?y\). Here \( \text{on} \) can have 2 variations, \( \text{on}(?b1,?b2) \) and \( \text{on}(?b2,?b1) \). Hence we consider these two as different lifted predicates. The number of such predicates is denoted as \( |P^*| \).

Table 1: Table showing the number of queries \( |Q| \) asked to the agent in our approach, as opposed to \( |Q| \) asked in the brute force solution having \( |\mathcal{M}| \) possible models to start with. Time shown is the average time taken per query for 5 runs of the agent interrogation algorithm.

| Domain   | \(|P^*|\) | \(|A|\) | \(|\mathcal{M}|\) | \(|Q|\) | Time/\(\text{sec}\) |
|----------|-----------|---------|----------------|--------|----------|
| gripper  | 5         | 3       | \(9^{18}\)     | 15 * 2^2 | 34       | 0.14      |
| blocks   | 9         | 4       | \(9^{36}\)     | 36 * 2^3 | 85       | 2.56      |
| elevator | 10        | 4       | \(9^{10}\)     | 40 * 2^10| 78       | 6.10      |
| logistics| 11        | 6       | \(9^{36}\)     | 66 * 2^11| 48       | 10.99     |
| parking  | 18        | 4       | \(9^{2}\)      | 72 * 2^18| 144      | 8.13      |
| satellite| 17        | 5       | \(9^{69}\)     | 85 * 2^17| 72       | 14.35     |
| stacks   | 10        | 12      | \(9^{120}\)    | 120 * 2^10| 122      | 10.69     |

Figure 5: Number of queries required and the average query computation time for the seven IPC domains. The domains are arranged in increasing order of \( |P^*| \times |A| \) from left to right.
lead to small number of queries asked.

5 Conclusion

In this paper we presented a novel approach for estimating the model of a black box autonomous agent by interrogating it. 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 number of objects in the domain and that 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.

Acknowledgements

We thank Abhyudaya Srinet for his help with the implementation. This work was supported in part by the NSF under grants IIS 1844325 and IIS 1909370.

References

[Aineto et al. 2019] 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, Jiménez, and Onaindia 2018] 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 1986] Alterman, R. 1986. An adaptive planner. In Proceedings of the Fifth AAAI National Conference on Artificial Intelligence, AAAI’86, 65–69. AAAI Press.

[Amir and Chang 2008] Amir, E., and Chang, A. 2008. Learning partially observable deterministic action models. Journal of Artificial Intelligence Research 33:349–402.

[Amir and Russell 2003] Amir, E., and Russell, S. 2003. Logical filtering. In Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-03, volume 3, 75–82.

[Bäckström and Jonsson 2013] 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.

[Camacho and McIlraith 2019] Camacho, A., and McIlraith, S. A. 2019. Learning interpretable models expressed in linear temporal logic. In International Conference on Automated Planning and Scheduling. ICAPS.

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

[Cresswell, McCluskey, and West 2009] 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 and Nilsson 1971] 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 and Walsh 1992] Giunchiglia, F., and Walsh, T. 1992. A theory of abstraction. Artificial intelligence 57(2-3):323–389.

[Helmert et al. 2017] 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 and Roth 1996] Khardon, R., and Roth, D. 1996. Reasoning with models. Artif. Intell. 87(1-2):187–213.

[Sacerdoti 1974] Sacerdoti, E. D. 1974. Planning in a hierarchy of abstraction spaces. Artificial intelligence 5(2):115–135.

[Srivastava, Russell, and Pinto 2016] Srivastava, S.; Russell, S.; and Pinto, A. 2016. Metaphysics of planning domain descriptions. In AAAI Conference on Artificial Intelligence.

[Stern and Juba 2017] 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, Wu, and Jiang 2007] 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.