Learning Causally Accurate Models for Autonomous Assessment of Deterministic Black-Box Agents

Pulkit Verma VERMA.PULKIT@ASU.EDU
Siddharth Srivastava SIDDHARTHS@ASU.EDU
Autonomous Agents and Intelligent Robots Lab,
School of Computing and Augmented Intelligence,
Arizona State University, Tempe, AZ 85281, USA

Abstract

This paper develops a new approach for estimating an interpretable, relational, and causally accurate model of a black-box autonomous agent that can plan and act in fully observable deterministic settings. Our main contributions are a new paradigm for estimating such models using a rudimentary query-response interface with the agent and a hierarchical querying algorithm that generates an interrogation policy. We also introduce dynamic causal decision networks (DCDNs) that capture the causal structure of planning models expressed in STRIPS-like languages. We show that the models we learn can be represented in the form of these DCDNs, and are causally accurate. Empirical evaluation of our approach shows that despite the exponential number of possible agent models in terms of the number of predicates and agent capabilities, our approach results in the correct and scalable estimation of interpretable agent models for a wide class of black-box autonomous agents. Our results also show that this approach can use predicate classifiers to learn interpretable models of planning agents that represent states as images.

1. Introduction

AI systems are being deployed at a faster rate than ever before. Many of these systems are intended to perform tasks specified by users who are not experts in AI. This leads to a pervasive problem for these users: how would they ascertain whether an AI system can perform a task safely in their home? Furthermore, most non-experts hesitate to ask questions about new AI tools (Mou & Xu, 2017) and often do not know which questions to ask for assessing the safe limits and capabilities of an AI system. This problem becomes even more challenging when we consider the fact that many of these AI systems have become so complex that even AI experts don't know how these systems work internally, and resort to post-hoc explanations of their decisions.

In this paper, we present an approach for addressing these issues. We propose a personalized agent-assessment module (AAM), shown in Fig. 1, which can be connected with an arbitrary AI agent using a simple interface. The desiderata for such an interface for third-party assessment of AI systems should be: (1) *interpretability*, the assessment system should be able to compute a user-interpretable model of the black-box AI system's capabilities; (2) *correctness*, the model computed by the assessment system should be accurate; (3) *generalizability*, the assessment system should be independent of the internals of the black-box AI system and hence should be able to work with a wide variety of AI systems; and (4) *minimal requirements*, as the assessment system is intended to be used by people who

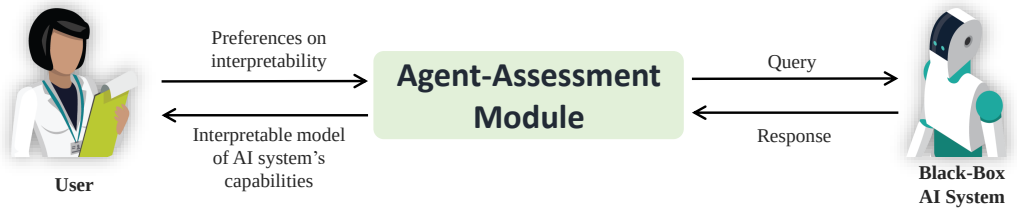

Figure 1: The agent-assessment module uses its user's preferred vocabulary, queries the AI system, and delivers a user-interpretable correct causal model of the AI system's capabilities. The AI system does not need to know the user's vocabulary or modeling language.

have not developed the AI systems being assessed, it should not place strong requirements on the design or implementation of the black-box AI systems to carry out the assessment.

To support these desiderata we leverage the framework of planning domain descriptions to learn and express symbolic models of the AI system in a relational STRIPS-like language (Fikes & Nilsson, 1971; Fox & Long, 2003). This representation feature conjunctive preconditions, add lists, and delete lists for each of the agent's capability, and can be easily translated into interpretable descriptions such as "in situations where X holds, if the agent executes capabilities $c_1, \ldots, c_k$ it would result in Y", where X and Y are in the user's vocabulary (Camacho & McIlraith, 2019). Such representations have been shown to be intuitive for humans in understanding deliberative behaviors of other agents (Malle, 2004; Miller, 2019). Furthermore, such models can be used to investigate interventions and support assessments of causality (Halpern, 2016).

We require the AI system to have only a rudimentary query-response interface using which it can answer simple queries. Consider a situation where a H(uman) ($\mathcal{H}$) wants a grocery-delivery robot ($\mathcal{A}$) to bring some groceries, but s/he is unsure whether it is up to the task and wishes to estimate $\mathcal{A}$'s internal model in an interpretable representation that s/he is comfortable with. If $\mathcal{H}$ was dealing with a delivery person, s/he might ask them questions such as "would you pick up orders from multiple persons?" and "do you think it would be alright to bring refrigerated items in a regular bag?" If the answers are "yes" during summer, it would be a cause for concern. Naïve approaches for generating such questions to ascertain the limits and capabilities of an agent are infeasible.[1]

The assessment module connects $\mathcal{A}$ with a simulator and provides a sequence of instructions, or a plan as a *query*. $\mathcal{A}$'s plan is executed in the simulator and the assessment module uses the simulated outcome as the response to the query. Thus, given an agent, the assessment module uses as input: a user-defined vocabulary, the agent's instruction set, and a compatible simulator. These inputs reflect natural requirements of the task and are already commonly supported: AI systems are already designed and tested using compatible simulators, and they need to specify their instruction sets in order to be usable. The user provides the predicates or concepts that they can understand and these concepts can be defined as functions on simulator states. Such concepts can be acquired through orthog-

1. Just 2 capabilities and 5 grounded propositions would yield $7^{2\times5} \sim 10^8$ possible STRIPS-like models – each proposition could be absent, positive or negative in the precondition and effects of each capability, and cannot be positive (or negative) in both preconditions and effect simultaneously. A query strategy that inquires about each occurrence of each proposition would be not only unscalable but also inapplicable to simulator-based agents that do not know their capabilities' preconditions and effects.

onal research on interactive concept acquisition (Kim, Shah, & Doshi-Velez, 2015; Lage & Doshi-Velez, 2020) or explained to users through demonstrations and training (Schulze et al., 2000). These concepts can be modeled as binary-valued *predicates* that have their associated evaluation functions (Mao et al., 2022).

This fundamental framework (Sec. 4) can be developed to support different types of agents as well as various query and response modalities. E.g., queries and responses could use a speech interface for greater accessibility, and agents with reliable inbuilt simulators/look-ahead models may not need external simulators. This would allow the assessment module to pose queries such as "what do you think would happen if you did $\langle$*query plan*$\rangle$", and the learned model would reflect $\mathcal{A}$'s self-assessment. The "agent" could be an arbitrary entity, although the expressiveness of the user-interpretable vocabulary would govern the scope of the learned models and their accuracy.

We have addressed this issue in the past using active interrogation based assessment of black-box AI systems (Verma, Marpally, & Srivastava, 2021; Nayyar, Verma, & Srivastava, 2022; Verma, Marpally, & Srivastava, 2022; Verma, Karia, & Srivastava, 2023) in a variety of settings. In contrast to these works, this paper (i) formally defines the interface requirements for a third-party assessment, (ii) performs an extensive complexity analysis, (iii) defines causal accuracy for planning models, and (iv) shows that the models learned using our approach are causally accurate.

Main contributions The main contributions of this work are: (i) first approach to formally define interface requirements for third-party assessment; (ii) formally define the agent assessment task; (iii) formally define causal soundness and completeness of planning models; (iv) propose an algorithm to solve the agent assessment task; and (v) show empirically that our algorithm learns causally accurate models, more efficiently than the baseline.

Our empirical evaluation (Sec. 6) shows that this method can efficiently learn correct models for black-box versions of agents using hidden models from the International Planning Competition (IPC)[2]. It also shows that the agent assessment module can use image-based predicate classifiers to infer correct models for simulator-based agents that respond with an image representing the result of query plan's execution.

2. Background

Models learned by the agent assessment module are in the form of planning models and we show in this work that these models are causally accurate (Sec. 5.2). In this section, we briefly discuss the formalism of the planning models, and outline the relevant background concepts for modeling causal relationships.

2.1 Planning Models

The agent assessment module assumes that the user needs to estimate the agent's model as a STRIPS-like planning model. Such models express the description of agent's high-level capabilities. In the literature (Verma et al., 2022, 2023), "actions" are used to refer to the core *functionality* of the agent, denoting the agent's decision choices, or primitive actions or controls that the agent could execute. In contrast, "capabilities" are used to refer to

2. https://www.icaps-conference.org/competitions

the *high-level behaviors* that the agent can perform using its AI algorithms for behavior synthesis, including planning and learning.

In this work, we assume that the agent provides its list of capabilities as input, and hence, in the rest of the paper, we use the term actions and capabilities interchangeably. Now, STRIPS-like planning models are formally defined as:

Definition 1. *A* **planning model** *is a tuple* $M = \langle P, C \rangle$*, where* $P = \{p_1^{r_1}, \ldots, p_n^{r_n}\}$ *is a finite set of predicates with arities* $r_i, i \in [1, n]$*;* $A = \{a_1, \ldots, a_k\}$ *is a finite set of parameterized actions (capabilities). Each action* $a_j \in A$ *is represented as a 3-tuple* $\langle \text{header}(a_j), \text{pre}(a_j), \text{eff}(a_j) \rangle$*, where* $\text{header}(a_j)$ *is the action header consisting of actions name and action parameters,* $\text{pre}(a_j)$ *represents the set of positive and negative predicate atoms that must be true or false, respectively in a state where* a_j *can be applied,* $\text{eff}(a_j)$ *is the set of positive or negative predicate atoms that will change to true or false, respectively as a result of execution of* a_j*.*

In the rest of the paper, we use the term "model" to refer to planning models. Given a model M and a set of objects O, let $S_{M,O}$ be the space of all states defined as maximally consistent sets of literals over the predicate vocabulary of M with O as the set of objects. We omit the subscript when it is clear from the context. An action $a \in A$ is applicable in a state $s \in S$ if $s \models \mathit{pre}(a)$. The result of executing s is a state $a(s) = s' \in S$ such that $s' \models \mathit{eff}(a)$, and all atoms not in $\mathit{eff}(a)$ have literal forms as in s. We extend this notation to express the result of executing a plan $\pi = \langle a_1, a_2, \ldots, a_n \rangle$ in a state s, i.e., $a_n(\ldots a_2(a_1(s))\ldots) = s_n$ as $\pi(s) = s_n$.

Lifted instantiated predicate Each predicate can be instantiated using the parameters of an action. The number of action parameters is bounded by the maximum arity of the action. E.g., consider the action `load_truck(?v1, ?v2, ?v3)` and predicate `(at ?x ?y)` in the IPC Logistics domain. The predicate `(at ?x ?y)` can be instantiated using action parameters `?v1`, `?v2`, and `?v3` as `(at ?v1 ?v1)`, `(at ?v1 ?v2)`, `(at ?v1 ?v3)`, `(at ?v2 ?v2)`, `(at ?v2 ?v1)`, `(at ?v2 ?v3)`, `(at ?v3 ?v3)`, `(at ?v3 ?v1)`, and `(at ?v3 ?v2)`. We represent the set of all such possible predicates instantiated with action parameters as lifted instantiated predicates P^*.

2.2 Observations

We compare our approach to the class of model learners that use the observations generated by the agent to learn the agent model. Such observations are defined as:

Definition 2. *Given a state space* S*, and a set of actions* A*, an* **observation trace** o *is an alternating sequence of states and actions of the form* $\langle s_0, a_1, s_1, a_2, \ldots, s_{n-1}, a_n, s_n \rangle$ *such that* $s_i \in S$*,* $a_i \in A$*, and* $\forall i \in [1, n]\ a_i(s_{i-1}) = s_i$*.*

The states s_{i-1} and s_i are called pre- and post-states of action a_i, respectively.

3. Formal Framework

As noted in introduction, the agent assessment module uses the following information as input: (i) the instruction set from the agent in the form of $\mathit{header}(a)$ for each $a \in A$; and

(ii) a predicate vocabulary P from the user with functional definitions of each predicate. This gives the assessment module sufficient information to perform a dialog with $\mathcal{A}$ about the outcomes of hypothetical action sequences. This dialog is performed in terms of queries and responses.

3.1 Form of Agent Queries

As mentioned earlier, the assessment module poses queries to the agent and based on $\mathcal{A}$'s responses θ it infers $\mathcal{A}$'s agent model. We express queries as functions that map models to answers.

Definition 3. *Given a set of predicates P and a set A of actions, let $\mathcal{U}$ be the set of all possible planning models (ref. Def. 1) expressible using P and A. Let Θ be the set of possible responses. A* **query** *q is a function $q : \mathcal{U} \to \Theta$.*

In this paper, we utilize only one class of queries: *plan outcome queries* (Q_{PO}), which are parameterized by a state s_I and a plan π. Let P^* be the set of predicates P instantiated with objects O in an environment. Q_{PO} queries ask $\mathcal{A}$ the length of the longest prefix of the plan π that it can execute successfully when starting in the state $s_I \subseteq P^*$ as well as the final state $s_F \subseteq P^*$ that this execution leads to. E.g., "Given that the truck `t1` and package `p1` are at location `l1`, what would happen if you executed the plan ⟨`load_truck(p1,t1,l1)`, `drive(t1,l1,l2)`, `unload_truck(p1,t1,l2)`⟩?"

A response to such queries can be of the form "I can execute the plan till step ℓ and at the end of it `p1` is in truck `t1` which is at location `l1`". Formally, the response θ_{PO} for plan outcome queries is a tuple $\langle \ell, s_F \rangle$, where ℓ is the number of steps for which the plan π could be executed, and $s_F \subseteq P^*$ is the final state after executing ℓ steps of the plan. If the plan π cannot be executed fully according to the agent model $M^{\mathcal{A}}$ then $\ell < len(\pi)$, otherwise $\ell = len(\pi)$. The final state $s_F \subseteq P^*$ is such that $M^{\mathcal{A}} \models \pi[1:\ell](s_I) = s_F$, i.e., starting with a state s_I, $M^{\mathcal{A}}$ successfully executed first ℓ steps of the plan π. Thus, $Q_{PO} : \mathcal{U} \to \mathbb{N} \times 2^P$, where $\mathbb{N}$ is the set of natural numbers.

3.2 Requirements for Independent Assessment

The requirements in an AI agent might change depending on the type of queries the agent is capable of answering. This is because we define the set of requirements as a function of the agent. Formally, we define the requirements on an agent as:

Definition 4. *Given a query class Q, with an associated response set Θ, the* **assessment requirement** *$\rho_{\mathcal{A}}$ on an autonomous agent $\mathcal{A}$ is a relation between Q and Θ, and is represented as $\rho_{\mathcal{A}}\langle Q, \Theta \rangle$.*

Given a plan-outcome query $q = \langle s_0, \pi \rangle$, where $\pi = \langle a_1, \ldots, a_n \rangle$, an autonomous agent $\mathcal{A}$ is said to support the set of requirements $\rho_{\mathcal{A}}$ if its response $\theta = \langle \ell, s_\ell \rangle$ satisfies:

$$\begin{aligned} \rho_{\mathcal{A}}(\langle s_0, \langle a_1, \ldots, a_n \rangle \rangle, \langle \ell, s_\ell \rangle) \triangleq\ & \ell < n \wedge \\ & \forall i \in 1, \ldots, \ell - 1,\ \exists s_i\ \mathcal{A} \models a_i(s_{i-1}) = s_i \wedge \mathcal{A} \models \neg(pre(a_{\ell+1}) \wedge s_\ell) \end{aligned} \tag{1}$$

We now define the overall problem of agent interrogation as follows. Given a class of queries and an agent with an unknown model supports the plan outcome query requirement above (1), determine the model of the agent. This can be formally defined as:

Definition 5. *An* **agent assessment task** *is defined as a quadruple* $\langle \mathcal{A}, P, C_H, \rho_{\mathcal{A}} \rangle$*, where* $\mathcal{C}$ *is the agent being assessed,* Q *is the class of queries that can be posed to the agent by the assessment module,* P *and* A_H *are the sets of predicates and action headers that the assessment module uses based on inputs from* $\mathcal{H}$ *and* $\mathcal{A}$*, and* $\rho_{\mathcal{A}}$ *is the assessment requirement that* $\mathcal{A}$ *must satisfy.*

The objective of our solution to the the agent interrogation task is to derive $\mathcal{A}$'s agent model $M^{\mathcal{A}}$ using Q, P, and A_H. We now introduce a running example which we'll use throughout the paper.

Running Example Consider a driving robot with a single action `drive(?t ?s ?d)`, parameterized by the truck it drives, source location, and destination location. Assume that all locations are connected, hence the robot can drive between any two locations. The predicates available are `(at ?t ?loc)`, representing the location of a truck; and `(src_blue ?loc)`, representing the color of the source location. Instantiating `at` and `src_blue` with parameters of the action *drive* gives four instantiated predicates `(at ?t ?s)`, `(at ?t ?d)`, `(src_blue ?s)`, and `(src_blue ?d)`.

3.3 Distinguishability and Prunability

Not all queries are useful, as 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. More precisely, a query q is said to *distinguish* a pair of models M_i and M_j, denoted as $M_i \mathtt{l}^q M_j$, iff $q(M_i) \neq q(M_j)$.

Definition 6. *Two models* M_i *and* M_j *are said to be* **distinguishable***, denoted as* $M_i \mathtt{l} M_j$*, iff there exists a query* q *that can distinguish between them, i.e.,* $\exists q\ M_i \mathtt{l}^q M_j$*.*

Given a pair of abstract models, we wish to determine whether one of them can be pruned, i.e., whether there is a query for which at least one of their answers is inconsistent with the agent's answer. Since this is computationally expensive to determine, and we wish to reduce the number of queries made to the agent, we first evaluate whether the two models can be distinguished by any query, independent of consistency of their response with that of the agent. If the models are not distinguishable, it won't be possible to try to prune one of them under the given query class.

Next, we determine if at least one of the two distinguishable models is consistent with the agent. When comparing the responses of two models at different levels of abstraction, we must consider the fact that the agent's response may be at a different level of abstraction if the given pair of models is abstract. Taking this into account, we formally define what it means for an abstract model M_i's response to be consistent with that of agent model $M^{\mathcal{A}}$:

Definition 7. *Let* q *be a query such that* $M_i \mathtt{l}^q M_j$*;* $q(M_i) = \langle \ell^i, s^i \rangle$*,* $q(M_j) = \langle \ell^j, s^j \rangle$*, and* $q(M^{\mathcal{A}}) = \langle \ell^{\mathcal{A}}, s^{\mathcal{A}} \rangle$*.* M_i*'s response to* q *is said to be* **consistent** *with that of* $M^{\mathcal{A}}$*, i.e.,* $q(M^{\mathcal{A}}) \models q(M_i)$ *iff* $\ell^{\mathcal{A}} = len(\pi^q)$*,* $len(\pi^q) = \ell^i$ *and* $s^i \subseteq s^{\mathcal{A}}$*.*

Using this notion of consistency, we can now reason that given a set of distinguishable models M_i and M_j, and their responses in addition to the agent's response to the distinguishing query, the models are prunable if and only if exactly one of their responses is consistent with that of the agent. Formally, we define prunability as:

Definition 8. *Given an agent-interrogation task* $\langle M^{\mathcal{A}}, Q, P, A_H \rangle$*, two models* M_i *and* M_j *are* **prunable***, denoted as* $M_i \Diamond M_j$*, iff* $\exists q \in Q : M_i \wr^q M_j \wedge (q(M^{\mathcal{A}}) \models q(M_i) \wedge q(M^{\mathcal{A}}) \not\models q(M_j)) \vee (q(M^{\mathcal{A}}) \not\models q(M_i) \wedge q(M^{\mathcal{A}}) \models q(M_j))$.

3.4 Components of Agent Models

In order to formulate our solution approach, we consider a model M to be comprised of components called *palm* tuples of the form $\lambda = \langle p, a, l, m \rangle$, where p is an instantiated predicate from the vocabulary P^*; a is an action from the set of parameterized actions A, $l \in \{pre, eff\}$, and $m \in \{+, -, \emptyset\}$. For convenience, we use the 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 predicate λ_p appears in an action λ_a at a location λ_l as a true (false) literal when mode λ_m is positive (negative), and is absent when $\lambda_m = \emptyset$. 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 mode m, i.e., $\lambda_1 \sim \lambda_2 \Leftrightarrow (\lambda_{1_p} = \lambda_{2_p}) \wedge (\lambda_{1_a} = \lambda_{2_a}) \wedge (\lambda_{1_l} = \lambda_{2_l}) \wedge (\lambda_{1_m} \neq \lambda_{2_m})$.

We also define the notion of *pal* tuples which are represented a 3-tuple $\langle p, a, l \rangle$. Each pal tuple $\gamma = \langle p, a, l \rangle$ corresponds to three pal tuples λ_m, $m \in \{+, -, \emptyset\}$, such that $\gamma_p = \lambda_{m_p}$, $\gamma_a = \lambda_{m_a}$, and $\gamma_l = \lambda_{m_l}$. A mode assignment to a *pal* tuple $\gamma = \langle p, a, l \rangle$ can result in 3 palm tuple variants $\gamma^+ = \langle p, a, l, + \rangle$, $\gamma^- = \langle p, a, l, - \rangle$, and $\gamma^\emptyset = \langle p, a, l, \emptyset \rangle$.

For a model $M = \langle P, A \rangle$, the set of all possible palm tuples and pal tuples that can be generates using $p \in P$ and $a \in A$ are represented as Λ and Γ, respectively.

Example 1. *Based on the running example, possible pal tuples are:*

- $\langle$`(at ?t ?s)`, `drive(?t ?s ?d)`, `pre`$\rangle$
- $\langle$`(at ?t ?s)`, `drive(?t ?s ?d)`, `eff`$\rangle$
- $\langle$`(at ?t ?d)`, `drive(?t ?s ?d)`, `pre`$\rangle$
- $\langle$`(at ?t ?d)`, `drive(?t ?s ?d)`, `eff`$\rangle$
- $\langle$`(src_blue ?s)`, `drive(?t ?s ?d)`, `pre`$\rangle$
- $\langle$`(src_blue ?s)`, `drive(?t ?s ?d)`, `eff`$\rangle$
- $\langle$`(src_blue ?d)`, `drive(?t ?s ?d)`, `pre`$\rangle$
- $\langle$`(src_blue ?d)`, `drive(?t ?s ?d)`, `eff`$\rangle$

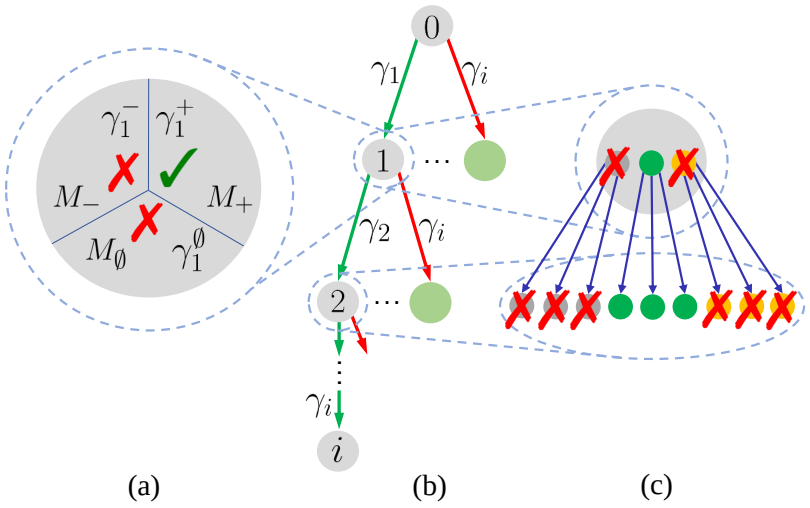

Figure 2: (b) Lattice segment explored in random order of $\gamma_i \in \Gamma$; (a) At each node, 3 abstract models are generated and 2 of them are discarded based on query responses; (c) An abstract model rejected at any level is equivalent to rejecting 3 models at the level below, 9 models two levels down, and so on.

3.5 Model Abstraction

We now define the notion of abstraction used in our solution approach. Several approaches have explored the use of abstraction in planning (Sacerdoti, 1974; Helmert, Haslum, & Hoffmann, 2007; Bäckström & Jonsson, 2013; Srivastava, Russell, & Pinto, 2016). The definition of abstraction used in this work extends the concept of predicate and propositional domain abstractions (Srivastava et al., 2016) to allow for the projection of a single *palm* tuple λ. An abstract model is one in which all variants of at least one pal tuple are absent. We define abstraction of a model as:

Definition 9. *Let Λ be the set of all possible palm tuples which can be generated using a predicate vocabulary P^* and an action header set A_H. Let U be the set of all consistent (abstract and concrete) models that can be expressed as subsets of Λ, such that no model has multiple variants of the same palm tuple. The* **abstraction of a model** *M with respect to a palm tuple $\lambda \in \Lambda$, is defined by $f_\lambda : U \to U$ as $f_\lambda(M) = M \setminus \lambda$.*

We extend this notation to define the abstraction of a set of models M with respect to a palm tuple λ as $X = \{f_\lambda(m) : m \in M\}$. We use this abstraction framework to define a subset-lattice over abstract models (Fig. 2(b)). Each node in the lattice represents a collection of possible abstract models which are possible variants of a pal tuple γ. E.g., in the node labeled 1 in Fig. 2(b), we have models corresponding to γ_1^+, γ_1^-, and $\gamma_1^\emptyset$. Two nodes in the lattice are at the same level of abstraction if they contain the same number of pal tuples. Two nodes n_i and n_j in the lattice are connected if all the models at n_i differ with all the models in n_j by a single palm tuple. As we move up in the lattice following these edges, we get more abstracted versions of the models, i.e., containing less number of pal tuples; and we get more concretized models, i.e., containing more number of pal tuples, as we move downward. We now define this model lattice:

Definition 10. *Let $\Lambda = \Gamma \times \{+, -, \emptyset\}$ be the set of all palm tuples. 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 mapping nodes to sets of abstract models, E is the set of lattice edges, and $\ell_E : E \to \Gamma$ is a labeling function mapping edges to pal tuples such that for each edge $n_i \to n_j$, $\ell_N(n_j) = \{\xi \cup \{\gamma^k\} | \, \xi \in \ell_N(n_i), \gamma = \ell_E(n_i \to n_j), k \in \{+, -, \emptyset\}\}$, and $\ell_N(\top) = \{\phi\}$ where $\top$ is the supremum containing the empty model ϕ.*

A node $n \in N$ in this lattice $\mathcal{L}$ can be uniquely identified by the sequence of pal tuples that label the edges leading to it from the supremum. As shown in Fig. 2(a), even though theoretically $\ell_N : N \to 2^{2^\Lambda}$, only three requirements of parent abstract models are stored. Additionally, in these model lattices, every node has an edge going out from it corresponding to each pal tuple that is not present in the paths leading to it from the most abstracted node. At any stage during the interrogation, nodes in such a lattice are used to represent the set of models consistent with the agent's responses up to that point. At every step, our algorithm creates queries that help us determine the next descending edge to take from a lattice node; corresponding to the path $0, 1, 2, \ldots, i$ in Fig. 2(b). This also avoids generating and storing the complete lattice, which can be doubly exponential in number of predicates and actions.

4. Solving the Agent Interrogation Task

Let Θ be the set of possible answers to queries. Thus, strings $\theta^* \in \Theta^*$ denote the information received by the assessment module at any point in the query process. Query policies for the agent interrogation task are functions $\Theta^* \to Q \cup \{Stop\}$ that map sequences of answers to the next query that the interrogator should ask. The process stops with the *Stop* query. In other words, for all answers $\theta \in \Theta$, all valid query policies map all sequences $x\theta$ to *Stop* whenever $x \in \Theta^*$ is mapped to *Stop*. This policy is computed and executed online.

We now discuss how we solve the agent interrogation task by incrementally adding palm variants to the class of abstract models and pruning out inconsistent models by generating distinguishing queries.

Example 2. *Consider the case of a delivery agent. Assume that AAM is considering two abstract models M_1 and M_2 having only one action* `load_truck(?package ?truck ?loc)` *and the predicates* `(at ?package ?loc)`, `(at ?truck ?loc)`, `(in ?package ?truck)`, *and that the agent's model is $M^{\mathcal{A}}$ (Fig. 1). AAM can ask the agent what will happen if $\mathcal{A}$ loads package* p1 *into truck* `t1` *at location* `l1` *twice. The agent would respond that it could execute the plan only till length* 1, *and the state at the time of this failure would be* `(at t1 l1)` $\wedge$ `(in p1 t1)`.

4.1 Agent Interrogation Algorithm

Algorithm 1 shows AAM's overall algorithm. It takes the agent $\mathcal{A}$, the set of instantiated predicates P^*, the set of all action headers A_H, and a set of random states S as input, and gives the set of functionally equivalent estimated models represented by *poss_models* as output. S can be generated in a preprocessing step given P^*. AIA initializes *poss_models* as a set consisting of the empty model ϕ (line 3) representing that AAM is starting at the supremum $\top$ of the model lattice.

Model	Precondition		Effect
$M^{\mathcal{A}}$	`(at ?truck ?loc)` $\wedge$ `(at ?package ?loc)`	$\rightarrow$	`(in ?package ?truck)` $\wedge \neg$`(at ?package ?loc)`
M_1	`(at ?truck ?loc)` $\wedge$ `(at ?package ?loc`	$\rightarrow$	`(in ?package ?truck)`
M_2	`(at ?truck ?loc)`	$\rightarrow$	`(in ?package ?truck)`
M_3	`(at ?truck ?loc)`	$\rightarrow$	`()`

Table 1: `load_truck(?package ?truck ?loc)` actions of the agent model $M^{\mathcal{A}}$ (unknown to $\mathcal{H}$) and three abstracted models M_1, M_2, and M_3.

In each iteration of the main loop (line 4), AIA maintains an abstraction lattice and keeps track of the current node in the lattice. It picks a pal tuple γ corresponding to one of the descending edges in the lattice from a node given by some input ordering of Γ. The correctness of the algorithm does not depend on this ordering. It then stores a temporary copy of *poss_models* as *new_models* (line 5) and initializes an empty set at each node to store the pruned models (line 6).

The inner loop (line 7) iterates over the set of all possible abstract models that AIA has not rejected yet, stored as *new_models*. It then loops over pairs of modes (line 8), which are later used to generate queries and refine models. For the chosen pair of modes, *generate_query()* is called (line 9) which returns two models concretized with the chosen modes and a query q which can distinguish between them based on their responses. Sec. 4.1.1 describes this process in detail.

AIA then calls *filter_models()* which poses the query q to the agent and the two models. Based on their responses, AIA prunes the models whose responses are not consistent with that of the agent (line 11). Then it updates the estimated set of possible models represented by *poss_models* (line 18). This process is explained in Sec. 4.1.2 in detail.

If AIA is unable to prune any model at a node (line 14), it modifies the pal tuple ordering (line 15). Sec. 4.1.3 explains this modification in detail. AIA continues this process until it reaches the most concretized node of the lattice (meaning all possible palm tuples $\lambda \in \Lambda$ are refined at this node). The remaining set of models represents the estimated set of models for $\mathcal{A}$. The number of resolved palm tuples can be used as a running estimate of the accuracy of the derived models. AIA requires $O(|P^*| \times |A|)$ queries as there are $2 \times |P^*| \times |A|$ pal tuples. However, our empirical studies show that we never generate so many queries. The rest of the section describes each of the submodules used in AIA.

4.1.1 Query Generation

The query generation process corresponds to the *generate_query()* module in AIA which takes a model M', the pal tuple γ, and 2 modes $i, j \in \{+, -, \emptyset\}$ as input; and returns the models $M_i = M' \cup \{\gamma^i\}$ and $M_j = M' \cup \{\gamma^j\}$, and a plan-outcome query q distinguishing them, i.e., $M_i \mathrel{\text{l}^q} M_j$.

Plan-outcome queries have two components, an initial state s_I, and a plan π. AIA gets s_I from the input set of random states $S \in \mathcal{S}$ (line 4). Using s_I as the initial state, the idea is to find a plan, which when executed by M_i and M_j will lead them either to different states or to a state where only one of them can execute the plan further. Later we pose the same query to $\mathcal{A}$ and prune at least one of M_i and M_j.

Algorithm 1 Agent Interrogation Algorithm (AIA)

Input: $\mathcal{A}, A_H, P^*, S$
Output: *poss_models*
Initialize $poss\_models = \{\phi\}$
for γ in some input pal ordering Γ **do**
  $new\_models \leftarrow poss\_models$
  $pruned\_models = \{\}$
  for each M' in *new_models* **do**
    for each pair $\{i, j\}$ in $\{+, -, \emptyset\}$ **do**
      q, M_i, $M_j \leftarrow generate\_query(M', i, j, \gamma, S)$
      $M_{prune} \leftarrow filter\_models(q, M^{\mathcal{A}}, M_i, M_j)$
      $pruned\_models \leftarrow pruned\_models \cup M_{prune}$
    end for
  end for
  if *pruned_models* is $\emptyset$ **then**
    $update\_pal\_ordering(\Gamma, S)$
    continue
  end if
  $poss\_models \leftarrow new\_models \times \{\gamma^+, \gamma^-, \gamma^\emptyset\} \setminus pruned\_models$
end for

Model	Action	Precondition		Effect
$M^{\mathcal{A}}$	`unload(?package ?truck ?loc)`	`(at ?truck ?loc)` ∧ `(in ?package ?truck)`	→	`(at ?package ?loc)` ∧ ¬`(in ?package ?truck)`
	`load(?package ?truck ?loc)`	`(at ?truck ?loc)` ∧ `(at ?package ?loc)`	→	¬`(at ?package ?loc)` ∧ `(in ?package ?truck)`
M_1	`unload(?package ?truck ?loc)`	¬`(in ?package ?truck)`	→	`()`
	`load(?package ?truck ?loc)`	`()`	→	`()`
M_2	`unload(?package ?truck ?loc)`	`(in ?package ?truck)`	→	`()`
	`load(?package ?truck ?loc)`	`()`	→	`()`
M_1 + $\texttt{in}_u$	`unload(?package ?truck ?loc)`	(`(in`$_u$ `?package ?truck)` ∨ ¬`(in ?package ?truck)`)	→	`(in`$_u$ `?package ?truck)`
	`load(?package ?truck ?loc)`	`(in`$_u$ `?package ?truck)`	→	`(in`$_u$ `?package ?truck)`
M_2 + $\texttt{in}_u$	`unload(?package ?truck ?loc)`	(`(in`$_u$ `?package ?truck)` ∨ `(in ?package ?truck)`)	→	`(in`$_u$ `?package ?truck)`
	`load(?package ?truck ?loc)`	`(in`$_u$ `?package ?truck)`	→	`(in`$_u$ `?package ?truck)`

Table 2: `load(?package ?truck ?loc)` and `unload(?package ?truck ?loc)` actions of the agent model $M^{\mathcal{A}}$ (unknown to $\mathcal{H}$) and two abstracted models M_1 and M_2, with and without the dummy predicate $\texttt{in}_u$.

Since the models M_i and M_j are abstract models, we need to ensure that we make accurate inferences based on their response to a query and compare it with $\mathcal{A}$'s response. Hence we cannot directly use M_i and M_j to ask questions. We illustrate this using the example below:

Example 3. *Consider an empty model ϕ with two actions* `unload(?package ?truck ?loc)` *and* `load(?package ?truck ?loc)`*. Now consider that it is being concretized with the pal tuple* $\gamma = \langle$`(in ?package ?truck)`, `unload(?package ?truck ?loc)`, pre$\rangle$*. The two models possible in this case are M_1 and M_2 shown in Tab. 2. It can be clearly seen that the model M_2 (without p_u) is an abstraction of $M^{\mathcal{A}}$. Now consider a query $q = \langle s_I, \pi \rangle$ where $s_I =$ {`(at p1 loc1)`, `(at t1 loc1)`}, and $\pi = \langle$`load(p1 t1 loc1)`, `unload(p1 t1 loc1)`$\rangle$. Now the response to this query by the agent $\mathcal{A}$ with model $M^{\mathcal{A}}$ in Tab. 2 will be $q(M^{\mathcal{A}}) = \langle 2,$ {`(at p1 loc1)`, `(at t1 loc1)`}$\rangle$. On the same query, the response of the abstract model M_1 will be $q(M_1) = \langle 2,$ {`(at p1 loc1)`, `(at t1 loc1)`}$\rangle$, whereas M_2's response will be $q(M_2) = \langle 1,$ {`(at p1 loc1)`, `(at t1 loc1)`}$\rangle$. Hence according to these responses, the M_2 will be discarded which is actually the correct model.*

This inconsistency happens because the model ϕ is only partially concretized in terms of the predicate `(in ?package ?truck)`, and this information is not captured in M_1 and M_2. To alleviate this issue, we add a predicate p_u to the models M_1 and M_2 as shown in Tab. 2. So p_u in any location (precondition or effect) helps capture the information that it is not known how a predicate p appears in that action's precondition (or effect). Without p_u, the planning problem can generate a plan as a query such that a model's response on it may be consistent with the agent even though the model is not an abstraction of $M^{\mathcal{A}}$. Now in the example above, with p_u added to the models M_1 and M_2, the response to the query for both the models will be $q(M_1) = q(M_2) = \langle 1,$ {`(at p1 loc1)`, `(at t1 loc1)`}$\rangle$, and hence q will no longer be a distinguishing query for these models. A possible distinguishing query in this case will be $q' = \langle s_I, \pi \rangle$ where $s_I =$ {`(in p1 t1)`, `(at t1 loc1)`}, and $\pi = \langle$`unload(p1 t1 loc1)`$\rangle$.

Generating plan outcome queries by reduction to planning We reduce the problem of generating a plan-outcome query from M_i and M_j to a planning problem. We add the pal tuple $\gamma = \langle p, a, l \rangle$ in modes i and j to M' to get M'_i and M'_j, respectively. If the location $l =$ *eff*, we add the palm tuple normally to M', i.e., $M'_m = M' \cup \langle p, a, l, m \rangle$, where $m \in \{i, j\}$. We modify these concretized models M'_j and M'_j further to reflect unknown effects on p as follows. Intuitively, an action makes an auxiliary predicate p_u (representing an unknown effect on p) true iff it does not have p in any mode as a precondition. To achieve this, we add the tuple $\langle p_u, a,$ *eff*$, + \rangle$ for all actions a that don't have p in any mode as an effect, and the tuple $\langle p_u, a,$ *eff*$, - \rangle$ for all other actions. Similarly, we further modify the model to reflect the unknown form of the preconditions in terms of p by adding p_u to the preconditions of all actions that do not have p in any mode in their precondition.

Note that p_u is added only for generating a distinguishing query and is not part of the models M_i and M_j returned by the query generation process.

We now show how to reduce plan-outcome query generation into a planning problem $\mathcal{P}(M''_i, M''_j)$ (line 5). $\mathcal{P}(M''_i, M''_j)$ uses conditional effects in its actions (in accordance with PDDL (McDermott et al., 1998)). The model used to define $\mathcal{P}(M''_i, M''_j)$ has predicates from both models M''_i and M''_j represented as $P^{M''_i}$ and $P^{M''_j}$ respectively, in addition to a new auxiliary 0-ary predicate p_ψ. The action headers are the same as A_H. Each action's precondition is a disjunction of the preconditions of M''_i and M''_j. This makes an action applicable in a state s if either M''_i or M''_j can execute it in s. The effect of each action has two conditional effects; the first applies the effects of both M''_i and M''_j's action

Algorithm 2 Query Generation Algorithm

Input: M', i, j, γ, S
Output: q, M_i, M_j
$M_i, M_j \leftarrow add\_palm(M', i, j, \gamma)$
for s_I in S **do**
  dom, $prob \leftarrow get\_planning\_prob\ (s_I, M_i, M_j)$
  $\pi \leftarrow planner(dom, prob)$
  $q \leftarrow \langle s_I, \pi \rangle$
  if π **then** break **end if**
end for
return q, $M' \cup \{\gamma^i\}$, $M' \cup \{\gamma^j\}$

if preconditions of both M_i'' and M_j'' are true, whereas the second makes the auxiliary predicate p_ψ true if precondition of only one of M_i'' and M_j'' is true. Note that p_ψ is initially false. Adding p_ψ helps identify the goal state $s_\mathcal{P}$.

Reason for adding $\mathbf{p}_\psi$ Formally, we express the planning problem $P_{PO}(M_i'', M_j'')$ as a 3-tuple $\langle M^{PO}, s_I, G \rangle$, where M^{PO} is a model with predicates $P^{PO} = \mathcal{P}^{M_i''} \cup \mathcal{P}^{M_j''} \cup \{p_\psi\}$, and actions A^{PO} where for each action $a \in A^{PO}$, $pre(a) = pre(a^{M_i''}) \vee pre(a^{M_j''})$, and

$$\begin{aligned} \mathit{eff}(a) = &(\mathit{when}\,(pre(a^{M_i''}) \wedge pre(a^{M_j''}))(\mathit{eff}(a^{M_i''}) \wedge \mathit{eff}(a^{M_j''}))) \\ &(\mathit{when}\,((pre(a^{M_i''}) \wedge \neg pre(a^{M_j''})) \vee (\neg pre(a^{M_i''}) \wedge pre(a^{M_j''})))\,(p_\psi)), \end{aligned}$$

The initial state $s_I = s_I^{M_i''} \wedge s_I^{M_j''}$, where $s_I^{M_i''}$ and $s_I^{M_j''}$ are copies of all predicates in s_I, and G is the goal formula expressed as $\exists p\ (p^{M_i''} \wedge \neg p^{M_j''}) \vee (\neg p^{M_i''} \wedge p^{M_j''}) \vee p_\psi$.

With this formulation, the goal is reached when an action in M_i'' and M_j'' differs in either a precondition (making only one of them executable in a state), or an effect (leading to different final states on applying the action). E.g., consider the models with differences in `load_truck(p1 t1 l1)` as shown in Fig. 1. From the state `(at t1 l1)` $\wedge$ `¬(at p1 l1)`, M_2 can execute `load_truck(p1 t1 l1)` but M_1 cannot. Similarly, in state `(at t1 l1)` $\wedge$ `(at p1 l1)`, executing `load_truck(p1 t1 l1)` will cause $M^\mathcal{A}$ and M_1 to end up in states differing in predicate `(at p1 l1)`. Hence, given the correct initial state, the solution to the planning problem P_{PO} will give the correct distinguishing plan.

We will formally prove in Sec. 5 that (i) this planning problem will always generate a solution query when using two models M_i'' and M_j'' that differ only in one palm tuple (Thm. 1), and (ii) any inferences about the consistency of models wrt. the agent model $M^\mathcal{A}$ based on responses to these queries are correct (Thm. 2).

4.1.2 Filtering Possible Models

This section describes the *filter_models()* module in Algorithm 1 which takes as input $M^\mathcal{A}$, M_i, M_j, and the query q (Sec. 4.1.1), and returns the subset M_{prune} which is not consistent with $M^\mathcal{A}$.

First, AAM *poses the query* q to M_i, M_j, and the agent $\mathcal{A}$. Based on the responses of all three, it determines if the two models are prunable, i.e., $M_i \Diamond M_j$. As mentioned in Def. 8, checking for prunability involves checking if the response 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 prunable, then the palm tuple being added in the inconsistent model cannot appear in any model consistent with $\mathcal{A}$. As we discard such palm tuples at abstract levels (as depicted in Fig. 2(a)), we prune out a large number of models down the lattice (as depicted in Fig. 2(c)), hence we keep the intractability of the approach in check and end up asking less number of queries.

4.1.3 Updating PAL ordering

This section describes the *update_pal_ordering()* module in AIA (line 15). It is called when the query generated by *generate_query()* module is not executable by $\mathcal{A}$, i.e., $len(\pi^q) \neq \ell^{\mathcal{A}}$. E.g., consider two abstract models M_2 and M_3 being considered by AAM (Fig. 1). At this level of abstraction, AAM does not have knowledge of the predicate `(at ?p ?l)`, hence it will generate a plan-outcome query with initial state {`(at ?t ?l)`} and plan ⟨`load_truck(p1 t1 l1)`⟩ to distinguish between M_2 and M_3. But this cannot be executed by the agent $\mathcal{A}$ as its precondition `(at ?p ?l)` is not satisfied, and hence we cannot discard any of the models.

Recall that in response to the plan-outcome query we get the failed action $a_F = \pi[\ell{+}1]$ and the final state s_F. Since the query plan π is generated using M_i and M_j (which differ only in the newly added palm tuple), they both would reach the same state $\overline{s}_F$ after executing first ℓ steps of π. Thus, we search S for a state $s \supset \overline{s}_F$ where $\mathcal{A}$ can execute a_F. Similar to Stern and Juba (2017), we infer that any predicate which is false in s will not appear in a_F's precondition in the positive mode. Next, we iterate through the set of predicates $p' \subseteq s \setminus \overline{s}_F$ and add them to $\overline{s}_F$ to check if $\mathcal{A}$ can still execute a_F. Thus, on adding a predicate $p \in p'$ to the state $\overline{s}_F$, if $\mathcal{A}$ cannot execute a_F, we add p in negative mode in a_F's precondition, otherwise in $\emptyset$ mode. All pal tuples whose modes are correctly inferred in this way are therefore removed from the pal ordering.

5. Formal Analysis of the AIA

In this section, we present a comprehensive theoretical analysis of AIA (Alg. 1). We show that AIA will always terminate and that the models returned by AIA are consistent with the agent's model and any model that is discarded by AIA is not consistent with that of the agent model. Then we analyse the causal accuracy of the learned model(s).

5.1 Theoretical Guarantees

In this section, we prove the main theorem of this paper (Thm. 3) which shows that the algorithm will terminate and the models returned by the algorithm are consistent with the agent's model. To prove it, we will prove that the approach prunes away models that are not consistent with the agent's model, and that the models returned by the algorithm are consistent with the agent's model. We will also show that the algorithm will terminate.

We will first show that a model with no pal tuples is consistent with the agent model according to Def. 7.

Lemma 1. *Consider an empty model M having 0 palm tuples, and that Alg. 1 concretizes M by adding a new pal tuple $\gamma = \langle p, a, l\rangle$ to M to generate models M'_i and M'_j, for $i, j \in \{+, -, \emptyset\}$. The planning problem $\mathcal{P}(M'_i, M'_j)$ has a solution if and only if $l =$ pre.*

Proof. We prove this in two parts. First, we show that if $l = pre$ then $\mathcal{P}(M'_i, M'_j)$ has a solution. Then, we show that if $l \neq pre$ ($l = \mathit{eff}$) then $\mathcal{P}(M'_i, M'_j)$ does not have any solution.

We prove the first part by construction. Recall that we add the pal tuple $\gamma = \langle p, a, l = pre\rangle$ in modes $i, j \in \{\emptyset, -, +\}$ to M to get M_i and M_j. Here $M = \{\}$, and pal tuple being added is $\gamma = \langle p, a, l = pre\rangle$ (line 9 of Alg. 1). Hence in the model for $\mathcal{P}(M'_i, M'_j)$, (i) precondition of all actions except a will be p_u; (ii) precondition of a will be $p \vee p_u$ in M_+, $\neg p \vee p_u$ in M_-, and $M_\emptyset$ will have empty precondition; and (iii) effects of all actions will be p_u. Now if $\{i, j\} \in \{\{+, \emptyset\}, \{+, -\}\}$ then $\mathcal{P}(M'_i, M'_j)$ has a solution if the initial state does not have p. Similarly if $\{i, j\} \in \{\{-, \emptyset\}, \{+, -\}\}$ then $\mathcal{P}(M'_i, M'_j)$ has a solution if the initial state has p. In both these cases, the solution plan π will be $\langle a\rangle$. Hence if $l = pre$ then $\mathcal{P}(M'_i, M'_j)$ has a solution.

Now we show that if $l \neq pre$ then $\mathcal{P}(M'_i, M'_j)$ does not have any solution. Here $M = \{\}$, and pal tuple being added is $\gamma = \langle p, a, l = \mathit{eff}\rangle$. Hence in the model for $\mathcal{P}(M'_i, M'_j)$, (i) effect of all actions except a will be p_u; (ii) effect of a will be p in M_+, $\neg p$ in M_-, and $M_\emptyset$ will have empty effect; and (iii) precondition of all actions will be p_u. Since p_u is not present in the initial state, no action is executable. So there is no plan possible; hence there is no solution for $\mathcal{P}(M'_i, M'_j)$ in this case. □

We will now formalize and show that the solution to the planning problem $P_{PO}(M_i, M_j)$ we just created is possible if the two models M_i and M_j have a distinguishing query. To formulate this, we will first define some additional notation. Consider the example 3 shown earlier. The initial state s_I used in P_{PO} will be $\texttt{(at}_i\ \texttt{t1 l1)} \wedge \texttt{(at}_j\ \texttt{t1 l1)}$, where $\texttt{(at}_x\ \texttt{t1 l1)}$ corresponds to `(at t1 l1)` predicate in the model M_x. We also represent the projection of a state s in planning problem P^{ij}_{PO} according to models M_i and M_j as $[s]_{M_i}$ and $[s]_{M_j}$, respectively. Here projection of a state s in planning problem P^{ij}_{PO} according to models M_i is the set of predicates with the subscript i, without their subscripts. E.g., consider if $s = \{\texttt{(at}_i\ \texttt{t1 l1)}, \texttt{(at}_j\ \texttt{p1 l1)}, p_\psi, \texttt{(at}_u\ \texttt{t1 l1)}\}$, then $[s]_{M_i} = \{\texttt{(at t1 l1)}\}$, and $[s]_{M_j} = \{\texttt{(at p1 l1)}\}$.

We now formalize an important property of the planning problem $P_{PO}(M_i, M_j)$'s solution. For brevity, we will represent $P_{PO}(M_i, M_j)$ as P^{ij} and the set of actions in P^{ij} as A^{ij}. We will show that the result of executing any action according to P^{ij} will be such that if $a^{ij}(s) = s'$ for any $a^{ij} \in A^{ij}$, then $a^x([s]_{M_x}) = [s']_{M_x}$, for all $x \in \{i, j\}$, where $\mathit{header}(a^{ij}) = \mathit{header}(a^x)$. This property is important to ensure that the responses to the queries by the models M_i and M_j are consistent with what was expected from the query generation process. Note that this *will not hold* for the final action in the plan if the action was executable according to only one of the models. This is because the final action will make the predicate p_ψ true in this case, whereas the model according to which the action was not executable will fail to execute the action, and the other model will make the correct effects true or false. We formalize this property as follows:

Lemma 2. *Consider a model M' that is an abstraction of the agent model $M^{\mathcal{A}}$, and M_i and M_j are two models concretized from M' such that they differ in mode of a single pal tuple $\langle p, a, l\rangle$. Consider $P^{ij}_{PO}(M_i, M_j)$ be a planning problem used to distinguish M_i and M_j with an initial state s_I, such that its solution is π, and $|\pi| = k$. If the result of executing any action $a^{ij} \in \pi$ according to P^{ij} will be such that if $a^{ij}_b(s) = s'$ for any $a^{ij}_b \in A^{ij}$ and $b \in [1, k-2]$, then $a^x([s]_{M_x}) = [s']_{M_x}$, for all $x \in \{i, j\}$, where $header(a^{ij}) = header(a^x)$.*

Proof. Recall that the goal to the planning problem $P^{ij}_{PO}(M_i, M_j)$ contains p_ψ in disjunction. So we will never execute an action in a state where p_ψ is true. Also recall that in the problem P^{ij}, $pre(a^{ij}) = pre(a^{M''_i}) \vee pre(a^{M''_j})$, where M''_x, $x \in \{i, j\}$ were the intermediate models used to create the planning problem P^{ij}. Now, $pre(a^{M''_x})$ cannot be false here, as $b \in [1, k-2]$, so we are considering all the actions other than the last action. So if $pre(a^{M''_x})$ was false for any a_b such that $b < k-1$, then that would've been the last action, which is not the case.

Now we will consider two cases: first, where we execute an action in a state without p_u, and second, in a state containing p_u. We analyze them one by one.

Case 1: Executing an action a in a state s, that does not have p_u predicates. This is the trivial case where the action a is executed when the precondition is satisfied according to at least one of the two models. Now when $pre(a^{M''_x})$ is true (the precondition corresponding to model M_x is satisfied), then the same precondition $pre(a)$ (projected version) is true in the actual model M_x too. This is because by construction the precondition can only have p_u predicates in addition to the normal predicates. Since projection removes p_u, if we project $pre(a^{M''_x})$, we'll get $pre(a)$ according to M_x. Hence Similar argument holds for the effect, hence the projection of effect $eff(a^{M''_x})$ will be same as $eff(a)$ according to M_x. Additionally, This implies that if $a_b(s) = s'$ for the case where p_u is false in s, the projection of states s and s' will be such that $a^x([s]_{M_x}) = [s']_{M_x}$.

Case 2: Executing an action a in a state s, that has p_u predicates. This condition means that the action is being executed in the state s where it is not known if the predicate p is true or false. Now, projecting s according to M_i (or M_j) would result in $[s]_{M_i}$ (or $[s]_{M_j}$) where p is not true. Similarly, p would also be absent from the precondition of a^i (or a^j). Hence if a was executable in s, a^i (or a^j) would be executable in $[s]_{M_i}$ (or $[s]_{M_j}$).

In both these cases, the effects of the action will change the states in an exact manner except for the p_u predicates. And since the projection of states according to a model anyway removes the p_u predicates, the resulting state s' according to the planning problem's action and $[s']_{M_i}$ ($[s']_{M_j}$) according to the model M_i (or M_j) will be the same. □

We will next show that the solution plan to $\mathcal{P}(M'_i, M'_j)$ always ends up with the action that is part of the pal tuple being concretized at that time. This will help us in limiting our analysis to, at most, the last two actions in the plan.

Lemma 3. *Let $M_i, M_j \in \{M_+, M_-, M_\emptyset\}$ be the models generated by adding pal tuple $\gamma = \langle p, a, l\rangle$ to M'. Suppose starting in a state s_I, π is a solution to $\mathcal{P}(M_i, M_j)$. The last action in the plan π will be a.*

Proof. We prove this by contradiction. Consider that the last action of the solution plan π is $a_F \neq a$, where a is the action in pal tuple $\gamma = \langle p, a, l \rangle$. Now, by construction, $\mathcal{P}(M_i, M_j)$ has a solution if after executing a_F (i) p_ψ is true, or (ii) $\exists p\ (p^{M''_i} \wedge \neg p^{M''_j}) \vee (\neg p^{M''_i} \wedge p^{M''_j})$ is true. Here M''_x are the intermediate models used in creating the planning problem $\mathcal{P}(M_i, M_j)$. We will now consider both the cases.

Case 1: p_ψ is true after executing a_F. As mentioned earlier, p_ψ becomes true when $((pre(a_F^{M''_i}) \wedge \neg pre(a_F^{M''_j})) \vee (\neg pre(a_F^{M''_i}) \wedge pre(a_F^{M''_j}))$. This means that when a_F is executed in a state s such that either $[s']_{M_i} \not\models pre(a_F^{M_i}) \wedge [s']_{M_j} \models pre(a_F^{M_j})$ or $[s']_{M_i} \models pre(a_F^{M_i}) \wedge [s']_{M_j} \not\models pre(a_F^{M_j})$. This means that $pre(a_F^{M_i}) \neq pre(a_F^{M_j})$. This is not possible as M_i and M_j are constructed from same model M' by making changes to action a. So $pre(a_F^{M_i})$ must be equal to $pre(a_F^{M_j})$. This means that our assumption that the last action of the plan must be $a_F \neq a$, must be false.

Case 2: $\exists p\ (p^{M''_i} \wedge \neg p^{M''_j}) \vee (\neg p^{M''_i} \wedge p^{M''_j})$ is true. This means that when a_F is executed in a state s such that $[s']_{M_i} \models pre(a_F^{M_i}) \wedge [s']_{M_j} \models pre(a_F^{M_j})$, their effects are not the same, i.e., $\mathit{eff}(a_F^{M_i}) \neq \mathit{eff}(a_F^{M_j})$. This is not possible as M_i and M_j are constructed from same model M' by making changes to action a. So $\mathit{eff}(a_F^{M_i})$ must be equal to $\mathit{eff}(a_F^{M_j})$. This means that our assumption that the last action of the plan must be $a_F \neq a$, must be false.

Since for both the cases, $a_F \neq a$ was false, hence $a_F = a$, i.e., the last action of the plan π will be a. □

We will next show that if the agent can execute the query successfully, then each intermediate state that each of the models generate on executing the plan in the query will be an abstraction of the states that the agent will generate as part of executing the same query. Additionally, all the intermediate states generated by both models will be identical.

Suppose $q = \langle s_I^q, \pi^q \rangle$ is a distinguishing query for two distinct models M_i, M_j, i.e., $M_i \mathrm{l}^q M_j$. Let $q_{1\ldots z}$ represent the query with same initial state s_I^q, and plan $\pi^q_{1\ldots z}$, i.e., first z ($z < len(\pi^q)$) actions from plan π^q.

Lemma 4. *Let $M_i, M_j \in \{M_+, M_-, M_\emptyset\}$ be the models generated by adding pal tuple γ to M' which is an abstraction of $M^{\mathcal{A}}$. Suppose $q = \langle s_I^q, \pi^q \rangle$ is a distinguishing query for two distinct models M_i, M_j, i.e., $M_i \mathrm{l}^q M_j$. Let the agent's response to the query $q(M^{\mathcal{A}})$ be $\langle \ell^{\mathcal{A}}, \langle p_1^{\mathcal{A}}, \ldots, p_k^{\mathcal{A}} \rangle \rangle$, where $\ell^{\mathcal{A}} = len(\pi^q)$. Let the response of models M_i, M_j, and $M^{\mathcal{A}}$ to the query $q_{1\ldots z}$ be: $q_{1\ldots z}(M_i) = \langle \ell^i, \langle p_1^i, \ldots, p_m^i \rangle \rangle$, $q_{1\ldots z}(M_j) = \langle \ell^j, \langle p_1^j, \ldots, p_n^j \rangle \rangle$, and $q_{1\ldots z}(M^{\mathcal{A}}) = \langle \ell^{\mathcal{A}}, \langle p_1^{\mathcal{A}}, \ldots, p_h^{\mathcal{A}} \rangle \rangle$. If $z \leq len(\pi^q) - 1$, then $m = n$, $\{p_1^i, \ldots, p_m^i\} = \{p_1^j, \ldots, p_n^j\}$ and $\{p_1^i, \ldots, p_m^i\} \setminus \{p_u\} \subseteq \{p_1^{\mathcal{A}}, \ldots, p_h^{\mathcal{A}}\}$.*

Proof. We first show that if the agent is able to execute the distinguishing query q successfully (i.e., $\ell^{\mathcal{A}} = len(\pi^q)$), then all the intermediate states generated by both the models while on executing the plan π^q starting in the initial state s^q are the same, and then show that the projection of these states are an abstraction of the corresponding intermediate state generated by the agent on executing the same query. Note that since q is a distinguishing query, $|\ell^i - \ell^j| \in \{0, 1\}$, also $z \leq \ell^i$ and $z \leq \ell^j$.

We now prove the first part by contradiction. The query q used to distinguish between M_i and M_j is generated using the planning problem P^{ij}. Suppose there exists $z' < len(\pi^q)$ for which the intermediate states generated after executing the plan $q_{1...z'}$ are not same according to M_i and M_j. Recall that P^{ij} has a solution if M_i and M_j have different preconditions or different effects for the same action (by definition of goal of P^{ij}). According to this, the plan $\pi^q_{1...z'}$ should also be a solution to P^{ij}. But this is not possible as any subsequence of π^q cannot be a solution otherwise π^q would never have been returned as the solution to P^{ij}. Hence all the intermediate states generated by both the models while on executing the plan π^q starting in the initial state s^q are the same.

Now we prove the second part that the intermediate states generated by both the models on executing the plan π^q starting in the initial state s^q are a subset of the corresponding projection of the intermediate state generated by the agent on executing the same query. Since M' is an abstraction of $M^{\mathcal{A}}$, all the palm tuples already present in it are also present in $M^{\mathcal{A}}$. The only new palm tuples in M_i and M_j are the ones involving pal tuple γ or involving predicate p_u. Now, as shown above, only the last action in the plan π^q will be a (corresponding to γ), and all the actions prior to that will not be a. Now, all the actions other than a, have the same preconditions and effects as they were in M' (which is an abstraction of $M^{\mathcal{A}}$), and since those actions in M' were consistent with that of $M^{\mathcal{A}}$, the effect of executing them will also be consistent with the agent $\mathcal{A}$. Thus, if s^q is the starting state, $z \leq len(\pi^q) - 1$, and $\pi^q_{1...z}(s) = s'_{ij}$ according to P^{ij}, and $\pi^q_{1...z}(s) = s'_{\mathcal{A}}$ according to $M^{\mathcal{A}}$, then $[s'_{ij}]_{M_i} \subseteq s'_{\mathcal{A}}$ and $[s'_{ij}]_{M_j} \subseteq s'_{\mathcal{A}}$. Note that in this notation, $[s'_{ij}]_{M_i} = \{p^i_1, \ldots, p^i_m\}$, $[s'_{ij}]_{M_j} = \{p^j_1, \ldots, p^j_n\}$, and $s'_{\mathcal{A}} = \{p^{\mathcal{A}}_1, \ldots, p^{\mathcal{A}}_h\}$.

□

Now we will see how these lemmas we proved so far combines to show that the planning problem created as part of Alg. 1 is guaranteed to generate a distinguishing query, if exists.

Theorem 1. *Consider a model M' that is an abstraction of the agent model $M^{\mathcal{A}}$, and M_i and M_j are two models concretized from M' such that they differ in mode of a single pal tuple. Given such pair of models M_i and M_j and an initial state s_I, the planning problem P^{ij}_{PO} has a solution iff M_i and M_j have a distinguishing plan-outcome query q_{PO}.*

Proof. We first show that if the planning problem P_{PO} has a solution, then M_i and M_j have a distinguishing plan-outcome query q_{PO} comprising of an initial state s_I and plan π^{PO}. By construction, the initial state s_I in q_{PO} and P^{ij} is same. Suppose P^{ij} has a solution plan π, which is a sequence of actions $a_1, a_2, \ldots, a_k$ where $a_x \in A_{P^{ij}}$, such that on executing this plan, the goal condition is met. Consider the trace of this execution be $\langle s_I, a_1, s_1, a_2, s_2, \ldots, s_{k-1}, a_k, s_G \rangle$. Recall that the goal of P^{ij} is that either the predicate p_ψ is true, or the final state according to the two models on executing the plan does not match. We will consider these cases individually.

Case 1: The predicate p_ψ is true on executing the plan, i.e., $s_G \models p_\psi$. By the construction of P^{ij}, it means that a_k made p_ψ true. It implies that only one of M_i's or M_j's preconditions were met in s_{k-1}. Consider that model whose preconditions were not met to be M_i. Now using lemma 4, we know that $[s_{k-1}]_{M_i} = [s_{k-1}]_{M_j}$, hence we will refer it as $[s_{k-1}]_{M_x}$ for brevity. Using Lemma 2, this also means that in the original models M_i and M_j, when

executing the corresponding action a_k^i (or a_k^j) in the state $[s_{k-1}]_{M_x}$, $[s_{k-1}]_{M_x} \not\models pre(a_k^i)$, and $[s_{k-1}]_{M_x} \models pre(a_k^j)$. Hence, if P^{ij} has a solution such that $s_G \models p_\psi$, then $M_i \wr M_j$.

Case 2: One of the predicates is true according to one of the model, and false according to another in the goal state, i.e., $s_G \models \exists p^i, p^j \in P^{ij}(p^i \wedge \neg p^j) \vee (\neg p^i \wedge p^j)$. By the construction of P^{ij}, it means that a_k made this condition true. It implies that both of M_i's or M_j's preconditions were met in s_{k-1}. Now using lemma 4, we know that $[s_{k-1}]_{M_i} = [s_{k-1}]_{M_j}$, hence we will refer it as $[s_{k-1}]_{M_x}$ for brevity. Using Lemma 2, this also means that in the original models M_i and M_j, when executing the corresponding action a_k^i (or a_k^j) in the state $[s_{k-1}]_{M_x}$, $[s_{k-1}]_{M_x} \models pre(a_k^i) \wedge pre(a_k^j)$. Since, in the predicate after executing a_k differs, $[s_k]_{M_i} \neq [s_k]_{M_j}$. Hence, if P^{ij} has a solution such that $s_G \models \exists p^i, p^j \in P^{ij}(p^i \wedge \neg p^j) \vee (\neg p^i \wedge p^j)$, then $M_i \wr M_j$.

□

Equivalent Models It is possible for AIA to encounter a pair of models M_i and M_j that are not prunable. In such cases, the models M_i and M_j are functionally equivalent and cannot be discarded. Hence, both the models end up in the set *poss_models* in line 18 of AIA.

We will next prove that a model is not an abstraction of the agent model if it is not consistent with that of the agent. But to prove that, we will use a couple of smaller results. We first start by showing that we can only prune an abstract model based on a query's responses if the agent can execute all actions in the query plan successfully.

Lemma 5. *Let $M_i \in \{M_+, M_-, M_\emptyset\}$ be the model generated by adding the pal tuple γ to M' which is an abstraction of the true agent model $M^{\mathcal{A}}$. Suppose q is a distinguishing query for two distinct models M_i and $M_j \in \{M_+, M_-, M_\emptyset\} \setminus M_i$. If $M^{\mathcal{A}}$ cannot execute all the actions in the query successfully, then we cannot decide consistency of the M_i (or M_j) response with that of the agent.*

Proof. Suppose $q = \langle s_I^q, \pi^q \rangle$ is a distinguishing query for two distinct models M_i, M_j, i.e. $M_i \wr^q M_j$, and the response of models M_i, M_j, and $M^{\mathcal{A}}$ to the query q are $q(M_i) = \langle \ell^i, \langle p_1^i, \ldots, p_m^i \rangle \rangle$, $q(M_j) = \langle \ell^j, \langle p_1^j, \ldots, p_n^j \rangle \rangle$, and $q(M^{\mathcal{A}}) = \langle \ell^{\mathcal{A}}, \langle p_1^{\mathcal{A}}, \ldots, p_k^{\mathcal{A}} \rangle \rangle$. We show that when $\ell^{\mathcal{A}} \neq len(\pi^q)$, i.e., $M^{\mathcal{A}}$ cannot execute all the actions in the query successfully, then we can make incorrect inferences about the consistency of the M_i (or M_j) response with that of the agent.

We prove this by counterexample. When $\ell^{\mathcal{A}} \neq len(\pi^q)$, consider the models M_1 with p_u, and M_2 with p_u in Tab. 2. Consider the initial state to be {`(in package1 truck1)`}, and the plan be $\langle$`unload(package1 truck1 location1)`$\rangle$. Now the responses of $M^{\mathcal{A}}$ and M_1 to this query will be $\langle 0, \{$`(in package1 truck1)`$\}\rangle$, whereas that of M_2 will be $\langle 1, \{p_u\}\rangle$. This happens because the first action in the plan failed for $M^{\mathcal{A}}$ because of the precondition `(at truck1 location1)` that is not satisfied in the initial state. On the other hand, the first action for M_1 failed because of the precondition $\neg$`(in package1 truck1)`. This happens because an action may execute successfully in an abstract model (M_2 here), but fail in the model which is an accurate concretization of it ($M^{\mathcal{A}}$ here) because of some predicate (`(at truck1 location1)` here) that the abstract models haven't added to their model. Hence M_i (or M_j) cannot be pruned if $\ell^{\mathcal{A}} \neq len(\pi^q)$. □

Lemma 6. *Let $M_i \in \{M_+, M_-, M_\emptyset\}$ be the model generated by adding the pal tuple γ to M' which is an abstraction of the true agent model $M^{\mathcal{A}}$. Suppose q is a distinguishing query for two distinct models M_i and $M_j \in \{M_+, M_-, M_\emptyset\} \setminus M_i$. If M_i's (or M_j's) response is not consistent with that of the agent, then it is not an abstraction of $M^{\mathcal{A}}$.*

Proof. Suppose $q = \langle s_I^q, \pi^q \rangle$ is a distinguishing query for two distinct models M_i, M_j, i.e. $M_i \mathrm{l}^q M_j$, and the response of models M_i, M_j, and $M^{\mathcal{A}}$ to the query q are $q(M_i) = \langle \ell^i, \langle p_1^i, \ldots, p_m^i \rangle \rangle$, $q(M_j) = \langle \ell^j, \langle p_1^j, \ldots, p_n^j \rangle \rangle$, and $q(M^{\mathcal{A}}) = \langle \ell^{\mathcal{A}}, \langle p_1^{\mathcal{A}}, \ldots, p_k^{\mathcal{A}} \rangle \rangle$. Now the model M_i's response to q is said to be *consistent* with that of $M^{\mathcal{A}}$ when $\ell^{\mathcal{A}} = len(\pi^q)$, $len(\pi^q) = \ell^i$ and $s^i \subseteq s^{\mathcal{A}}$, where $s^i = \{p_1^i, \ldots, p_m^i\} \setminus p_u$ and $s^{\mathcal{A}} = \{p_1^{\mathcal{A}}, \ldots, p_k^{\mathcal{A}}\}$. We prove this in multiple parts. We have already shown in Lemma 4 that $\ell^{\mathcal{A}} = len(\pi^q)$ is a necessary condition for consistency. We will now show that if either $len(\pi^q) = \ell^i$ or $s^i \subseteq s^{\mathcal{A}}$ are not true, then M_i is not an abstraction of $M^{\mathcal{A}}$.

We first show that if $len(\pi^q) \neq \ell^i$, then M_i is not an abstraction of $M^{\mathcal{A}}$. Since $\ell^{\mathcal{A}} = len(\pi^q)taa$, the agent can execute each of the actions in the plan. Now if M_i is also able to execute an action whereas M_j can (since q is a distinguishing query), then using Lemma 2 it can only be the last action. Now the set of preconditions for an abstracted model will never be larger than its corresponding concretized model, hence if an action is executable in the concretized model, it should also be executable in the abstracted model. Now since the other actions in the plan are correct as M' is an abstraction of $M^{\mathcal{A}}$, hence if $len(\pi^q) \neq \ell^i$, then M_i is not an abstraction of $M^{\mathcal{A}}$.

We now show that if $s^i \not\subseteq s^{\mathcal{A}}$, then M_i is not an abstraction of $M^{\mathcal{A}}$. Since $\ell^{\mathcal{A}} = len(\pi^q)$, the agent can execute each of the actions in the plan. Now if the states that M_i and M_j do not end up in the same state, then using Lemma 2 it can only be after the last action. Now the set of effects for an abstracted model will never be larger than its corresponding concretized model, hence if an action is executable in the concretized model and ends up in a state, then the abstracted model should also reach a state where only the subset of its effects are true. Hence if $s^i \not\subseteq s^{\mathcal{A}}$, then M_i is not an abstraction of $M^{\mathcal{A}}$. □

Theorem 2. *Let $M_i \in \{M_+, M_-, M_\emptyset\}$ be the model generated by adding the pal tuple γ to M' which is an abstraction of the true agent model $M^{\mathcal{A}}$. Suppose q is a distinguishing query for two distinct models M_i and $M_j \in \{M_+, M_-, M_\emptyset\} \setminus M_i$. If M_i (or M_j) is pruned out by Alg. 1, then it is not an abstraction of $M^{\mathcal{A}}$.*

Proof. We prove this by mathematical induction. Suppose $q = \langle s_I^q, \pi^q \rangle$ is a distinguishing query for two distinct models M_i, M_j, i.e. $M_i \mathrm{l}^q M_j$, and the response of models M_i, M_j, and $M^{\mathcal{A}}$ to the query q are $q(M_i) = \langle \ell^i, \langle p_1^i, \ldots, p_m^i \rangle \rangle$, $q(M_j) = \langle \ell^j, \langle p_1^j, \ldots, p_n^j \rangle \rangle$, and $q(M^{\mathcal{A}}) = \langle \ell^{\mathcal{A}}, \langle p_1^{\mathcal{A}}, \ldots, p_k^{\mathcal{A}} \rangle \rangle$. Now when $\ell^{\mathcal{A}} \neq len(\pi^q)$, none of M_i or M_j can be discarded as shown in Lemma 4.

When $\ell^{\mathcal{A}} = len(\pi^q)$, and M_i (or M_j) is pruned then it means that either $len(\pi^q) \neq \ell^i$ or $\{p_1^i, \ldots, p_m^i\} \not\subseteq \{p_1^{\mathcal{A}}, \ldots, p_k^{\mathcal{A}}\}$. So we will now prove that if $len(\pi^q) \neq \ell^i$ or $\{p_1^i, \ldots, p_m^i\} \setminus p_u \not\subseteq \{p_1^{\mathcal{A}}, \ldots, p_k^{\mathcal{A}}\}$ then M_i is not an abstraction of $M^{\mathcal{A}}$.

Let P(n) be the proposition that for every model with n pal tuples, which is consistent with $M^{\mathcal{A}}$, refining it with a pal tuple with the correct mode according to Def. 3 will prune out the models that are not an abstraction of $M^{\mathcal{A}}$.

Base Case: The proof for P(0) being true is by case analysis. Assume the model $M' = \{\}$, which is consistent with $M^{\mathcal{A}}$, is concretized with pal tuple $\gamma = \langle p, a, l \rangle$. There are only two cases possible as the location can only be a precondition or an effect.

Case 1: Consider $l = pre$. This case splits into 2 subcases, based on if the predicate will be true in the initial state or not. Note that the plan will have only one action as the models are completely empty except the only action that is being concretized.
Case 1.1: If $p \in s_I^q$, $\pi^q = \langle a \rangle$, and $\ell^{\mathcal{A}} = len(\pi^q) = 1$, then $\langle p, a, pre, - \rangle \notin M^{\mathcal{A}}$. Also $M_j \text{l}^q M_-$, where $j \in \{+, \emptyset\}$, as $\ell^j = 1$, and $\ell^- = 0$. Hence P(0) is true.

Case 1.2: If $\neg p \in s_I^q$, $\pi^q = \langle a \rangle$, and $\ell^{\mathcal{A}} = len(\pi^q) = 1$, then $\langle p, a, pre, + \rangle \notin M^{\mathcal{A}}$. Also $M_j \text{l}^q M_+$, where $j \in \{-, \emptyset\}$, as $\ell^j = 1$, and $\ell^+ = 0$. Hence P(0) is true.

Case 2: Consider $l = eff$. If $M = \{\}$ and $l = eff$, $\forall i, j \in \{+, -, \emptyset\}$, $\nexists q$ $M_i \text{l}^q M_j$ as shown in Lemma 1. Hence P(0) is true.

Inductive Step: Assume that P(n) is true for some $n \geq 0$; that is we have a model M', with n palm tuples, which is an abstraction of $M^{\mathcal{A}}$, and refining it with a pal tuple $\gamma = \langle p, a, l \rangle$ will generate models with $n + 1$ tuples. From Lemma 4, we know that before executing the last action, the state reached by both the abstracted models ($\overline{s}_{F-1}$) will be a subset of the state reached by $M^{\mathcal{A}}$ (s_{F-1}). There are two cases:

Case 1: Consider $l = pre$. Since $l = pre$, $p_u \notin \overline{s}_{F-1}$. This case splits into 2 subcases:

Case 1.1: If $p \in \overline{s}_{F-1}$, and $\ell^{\mathcal{A}} = len(\pi^q)$, then $\langle p, a, pre, - \rangle \notin M^{\mathcal{A}}$. Also $M_j \text{l}^q M_-$, where $j \in \{+, \emptyset\}$, as $\ell^j = len(\pi^q)$, and $\ell^- = len(\pi^q) - 1$. Thus, M_- is not an abstraction of $M^{\mathcal{A}}$. Hence P(n) is true.

Case 1.2: If $\neg p \in \overline{s}_{F-1}$, and $\ell^{\mathcal{A}} = len(\pi^q)$, then $\langle p, a, pre, + \rangle \notin M^{\mathcal{A}}$. Also $M_j \text{l}^q M_+$, where $j \in \{-, \emptyset\}$, as $\ell^j = len(\pi^q)$, $\ell^+ = len(\pi^q) - 1$. Thus, M_+ is not an abstraction of $M^{\mathcal{A}}$. Hence P(n) is true.

Case 2: Consider $l = eff$. Since $l = eff$, p_u may or may not be in $\overline{s}_{F-1}$. In either case, the full plan is executed in M_i, M_j and $M^{\mathcal{A}}$. Hence we can compare the states reached after executing the complete plan. Let $\overline{s}_F^{M_i} = \{p_1^i, \ldots, p_m^i\}$, $\overline{s}_F^{M_j} = \{p_1^i, \ldots, p_n^i\}$, and $s_F = \{p_1^i, \ldots, p_k^i\}$ be the final states reached upon executing π^q in M_i, M_j and $M^{\mathcal{A}}$ respectively and $\overline{s}_{F-1}$ is the state reached in M_i and M_j before executing action a. This case splits into 2 subcases:

Case 2.1: If $p \in \overline{s}_{F-1}$. If $p \in s_F$, $\langle p, a, eff, - \rangle \notin M^{\mathcal{A}}$ and $\overline{s}_F^{M_-} \not\subseteq s_F$. Similarly if $\neg p \in s_F$, $\langle p, a, eff, + \rangle \notin M^{\mathcal{A}}$ and $\overline{s}_F^{M_+} \not\subseteq s_F$, and $\langle p, a, eff, \emptyset \rangle \notin M^{\mathcal{A}}$ and $\overline{s}_F^{M_\emptyset} \not\subseteq s_F$. Hence P(n) is true.

Case 2.2: If $\neg p \in \overline{s}_{F-1}$. If $\neg p \in s_F$, $\langle p, a, eff, + \rangle \notin M^{\mathcal{A}}$ and $\overline{s}_F^{M_+} \not\subseteq s_F$. Similarly if $p \in s_F$, $\langle p, a, eff, - \rangle \notin M^{\mathcal{A}}$ and $\overline{s}_F^{M_-} \not\subseteq s_F$, and $\langle p, a, eff, \emptyset \rangle \notin M^{\mathcal{A}}$ and $\overline{s}_F^{M_\emptyset} \not\subseteq s_F$. Hence P(n) is true.

This proves that if we add a pal tuple to a model that is an abstraction of $M^{\mathcal{A}}$, then we prune only inconsistent models M_i whenever $len(\pi^q) \neq \ell^i$ or $\{p_1^i, \ldots, p_m^i\} \not\subseteq \{p_1^{\mathcal{A}}, \ldots, p_k^{\mathcal{A}}\}$ when $\ell^{\mathcal{A}} = len(\pi^q)$. □

We will now prove that the set of estimated models returned by AIA is correct and the returned models are functionally equivalent to the agent's model, and no correct model is discarded in the process. We will henceforth refer to Alg. 1 as AIA (Agent Interrogation Algorithm). To prove our next theorem we'll need some additional lemmas that we prove below. The first one mentions that AIA never prunes away a model whose possible concretization is an abstraction of the agent model, and the second one shows that AIA always terminates.

Lemma 7. *Given an agent $\mathcal{A}$ with a model $M^{\mathcal{A}}$, and an abstract model M^{abs}, if AIA prunes away an abstract model M^{abs}, then no possible concretization of M^{abs} will be an abstraction of the agent model $M^{\mathcal{A}}$.*

Proof. We prove this using simple inference. At each node in the lattice, we always prune away some of the models. If we discard an inconsistent model, it is because some palm tuple in the model has a different mode m, than that of $M^{\mathcal{A}}$ (Lemma 6). This incorrect palm tuple will also be present in all its concretizations, making all of them inconsistent with $M^{\mathcal{A}}$. Theorem 2 proves that at each node the models pruned away by AIA are not an abstraction of the agent model. This means that they have at least one of the pal tuples in a mode that does not match that of the agent model. Now concretizing such a model will only add other pal tuples in one of the three modes as explained in section 3.5, hence the incorrect mode of the pal tuple will remain unchanged thereby making all possible concretizations of such a model an incorrect abstraction of the agent model. □

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

Lemma 8. *The Agent Interrogation Algorithm (Alg. 1) will always terminate.*

Proof. As mentioned in Def. 16, in our subset lattice, "level" is equivalent to the number of refined pal tuples. At each step of the algorithm, when we consider a refinement in terms of pal tuples, we are left with one or more variants of the pal tuple. This ensures that we never refine the models more than once at a single level in the lattice. Since we refine at least one pal tuple in every iteration of the algorithm, the algorithm is bound to terminate as the number of pal tuples is finite for a finite number of propositions and actions under consideration. □

Theorem 3. *The Agent Interrogation Algorithm (Alg. 1) will always terminate and return a set of models, each of which are functionally equivalent to the agent's model $M^{\mathcal{A}}$.*

Proof. Theorem 1 and Theorem 2 prove that whenever we get a prunable query, AIA discards only the models that are not abstractions of the agent model, thereby ensuring that no model that is an abstraction of the agent model is discarded. When we do not get a prunable query, AIA infers the correct precondition(s) of the failed action using *update_pal_ordering()*, hence the number of refined palm tuples always increase with the

number of iterations of AIA (line 4 of AIA), thereby ensuring its termination in finite time. And when the algorithm terminates, the models that remain have all their responses consistent with that of the agent's model and hence are functionally equivalent to that of the agent model. □

We now explain how AIA models the causally accurate relationships of the domain.

5.2 Causal Accuracy of the Learned Models

We compare the properties of models learned by AIA with those of approaches that learn the models from observational data only. For the methods that learn models in STRIPS-like the learned models can be classified as causal, but it is not necessary that they are sound with respect to the ground truth model $M^{\mathcal{A}}$ of the agent $\mathcal{A}$. E.g., in case of the robot driver discussed earlier, these methods can learn a model where the precondition of the action drive is `src_blue` if all the observation traces that are provided to it as input had `src_blue` as true. This can happen if all the source locations are painted blue. To avoid such cases, some of these methods run a pre-processing or a post-processing step that removes all static predicates from the preconditions. However, if there is a paint action in the domain that changes the color of all source locations, then these ad-hoc solutions will not be able to handle that. Hence, these techniques may end up learning spurious preconditions as they do not have a way to distinguish between correlation and causations.

On the other hand, it is also not necessary that the models learned by approaches using only observational data are complete with respect to the ground truth model $M^{\mathcal{A}}$ of the agent $\mathcal{A}$. This is because they may miss to capture some causal relationships if the observations do not include all the possible transitions, or contains only the successful actions. E.g., if we have additional predicates `(city_from ?loc)`, and `(city_to ?loc)` in the domain, and all the observed transitions are for the transitions within same city, then the model will not be able to learn if the source city and destination city have to be same for driving a truck between them.

5.2.1 Causal Models

In this section, we provide an overview of terminology regarding causal implications from Halpern (2015). We will use this framework to show that models learned by our approach are causally accurate.

Definition 11. *A* **causal model** M *is defined as a 4-tuple* $\langle U, V, R, F\rangle$ *where* U *is a set of exogenous variables (whose values are determined by factors outside the model),* V *is a set of endogenous variables (whose values are directly or indirectly derived from the exogenous variables),* R *is a function that associates with every variable* $Y \in U \cup V$ *a nonempty set* $R(Y)$ *of possible values for* Y*, and* F *is a function that associates with each endogenous variable* $X \in V$ *a structural function denoted as* F_X *such that* F_X *maps* $\times_{Z\in(U\cup V-\{X\})}R(Z)$ *to* $R(X)$*.*

Note that the values of exogenous variables are not determined by the model; a setting $\vec{u}$ of values of exogenous variables is termed as a *context* by Halpern (2016). This helps in defining a causal setting as:

Definition 12. *Given a setting of exogenous variables $u \in U$, a* **causal setting** *is a pair $(M, \vec{u})$ consisting of a causal model M and context $\vec{u}$.*

A causal formula φ is true or false in a causal model, given a context. Hence, $(M, \vec{u}) \models \varphi$ if the causal formula φ is true in the causal setting $(M, \vec{u})$.

Every causal model M can be associated with a directed graph, $G(M)$, in which each random variable X is represented as a vertex and the causal relationships between these variables are represented as directed edges between members of $U \cup \{V \setminus X\}$ and X (Pearl, 2009). We use the term causal networks when referring to these graphs to avoid confusion with the notion of causal graphs used in the planning literature (Helmert, 2004).

To perform an analysis with interventions, we use *do-calculus* introduced in Pearl (1995). To perform interventions on a set of variables $X \in V$, do-calculus assigns values $\vec{x}$ to $\vec{X}$, and evaluates the effect using the causal model M. This is termed as *do*$(\vec{X} = \vec{x})$ action. To define this concept formally, we first define *submodels* (Pearl, 2009).

Definitions 13-16 are by Halpern (2016). These definitions summarize the concepts we use to define and assess the causal accuracy of the learned agent models.

Definition 13. *Let M be a causal model, X a set of variables in V, and $\vec{x}$ a particular realization of $\vec{X}$. A* **submodel** *$M_{\vec{x}}$ of M is the causal model $M_{\vec{x}} = \langle U, V, R, F^{\vec{x}} \rangle$ where $F^{\vec{x}}$ is obtained from F by setting $X' = x'$ (for each $X' \in \vec{X}$) instead of the corresponding $F_{X'}$, and setting $F^{\vec{x}}_Y = F_Y$ for each $Y \notin X$.*

We now define what it means to intervene $\vec{X} = \vec{x}$ using the action $do(\vec{X} = \vec{x})$. Let M be a causal model, X a set of variables in V, and $\vec{x}$ a particular realization of $\vec{X}$. The effect of action $do(\vec{X} = \vec{x})$ on M is given by the submodel $M_{\vec{x}}$.

In general, there can be uncertainty about the effects of these interventions, leading to probabilistic causal networks, but in this work, we work with fully observable and deterministic settings, hence assume that interventions do not lead to uncertain effects.

We can also derive the structure of causal networks using interventions in the real world, as interventions allow us to find if a variable Y depends on another variable X. We use Halpern (2016)'s notion of dependence as follows.

Definition 14. *A variable Y* **depends on** *a variable X if there is some setting of all the variables in $U \cup V \setminus \{X, Y\}$ such that varying the value of X in that setting results in a variation in the value of Y.*

We now use these concepts to define what a causal formula is (Halpern, 2016) and then use it to define what we mean by an actual cause.

Definition 15. *Given a signature $\mathcal{S} = (U, V, R)$, a* primitive event *is a formula of the form $X = x$, for $X \in V$ and $x = R(X)$. A* causal formula *is $[\vec{Y} \leftarrow \vec{y}]\varphi$, where φ is a Boolean combination of primitive events, $\vec{Y} = \langle Y_1, Y_2, \ldots Y_i \rangle$ are distinct variables in V, and $y_i \in R(Y_i)$.*

$[\vec{Y} \leftarrow \vec{y}]\varphi$ means that φ would hold if Y_k were set to y_k, for $k = 1, \ldots, i$. We next formally define an actual cause.

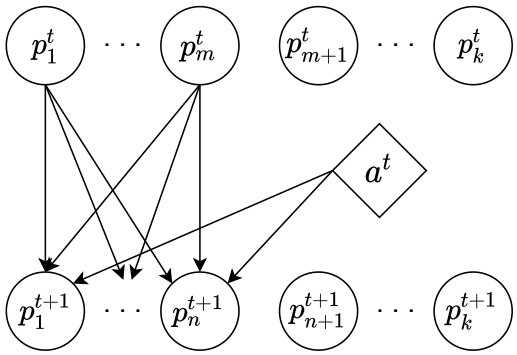

Figure 3: An example of a Dynamic Causal Decision Network (DCDN). p_i^t and p_i^{t+1} are the action-parameter instantiated predicates at time t and $t+1$ respectively and a_t is a decision node representing the decision to execute action the parameterized a at time t.

Definition 16. *Let $X \subseteq V$ be a subset of endogenous variables V, and φ be a boolean causal formula expressible using variables in V. $\vec{X} = \vec{x}$ is an* **actual cause** *of φ in the causal setting $(M, \vec{u})$, i.e., $(\vec{X} = \vec{x}) \overset{(M,\vec{u})}{\rightsquigarrow} \varphi$, if the following conditions hold:*

AC1. $(M, \vec{u}) \models (\vec{X} = \vec{x})$ and $(M, \vec{u}) \models \varphi$.

AC2. There is a set $\vec{W}$ of variables in V and a setting $\vec{x}'$ of the variables in $\vec{X}$ such that if $(M, \vec{u}) \models \vec{W} = \vec{w}^*$, then $(M, \vec{u}) \models [\vec{X} \leftarrow \vec{x}', \vec{W} \leftarrow \vec{w}^*]\neg\varphi$.

AC3. $\vec{X}$ is minimal; there is no strict subset $\vec{X}'$ of $\vec{X}$ such that $\vec{X}' = \vec{x}'$ satisfies conditions AC1 and AC2, where $\vec{x}'$ is the restriction of $\vec{x}$ to the variables in $\vec{X}$.

AC1 mentions that unless both φ and $\vec{X} = \vec{x}$ occur at the same time, φ cannot be caused by $\vec{X} = \vec{x}$. AC2[3] mentions that there exists a $\vec{x}'$ such that if we change a subset $\vec{X}$ of variables from some initial value $\vec{x}$ to $\vec{x}'$, keeping the value of other variables $\vec{W}$ fixed to $\vec{w}^*$, φ will also change. AC3 is a minimality condition which ensures that there are no spurious elements in $\vec{X}$.

In this section, we'll first show a mapping between the STRIPS-like models that we learn and the causal models defined earlier. And then we'll define the causal soundness and completeness of one causal model w.r.t. another causal model, and show that the models learned by Alg. 1 are causally accurate.

5.2.2 Representing Planning Models as Causal Networks

The classical causal model framework used in Def. 11 lacks the temporal elements and decision nodes needed to express the causal relationships in the planning models.

To express actions in the model, we use the decision nodes similar to Dynamic Decision Networks (Kanazawa & Dean, 1989). To express the temporal behavior of planning models, we use the notion of Dynamic Causal Models (Pearl, 2009) and Dynamic Causal Networks (DCNs) (Blondel, Arias, & Gavaldà, 2017). These are similar to causal models and causal

3. Halpern (2016) terms this version of AC2 as AC2(a^m)

networks respectively, with the only difference that the variables in these are time-indexed, allowing for analysis of temporal causal relations between the variables. We also introduce additional boolean variables to capture the executability of the actions. The resulting causal model is termed as a causal action model, and we express such models using a Dynamic Causal Decision Network (DCDN).

A general structure of a dynamic causal decision network is shown in Fig. 3. All the decision variables and the executability variables p_i^t, where $i \in [0, k]$, where k is the number of instantiated predicates, in a domain are endogenous. There is an edge from each predicate in an action's precondition to each predicate in an action's effect. All the variables are endogenous because we can perform interventions on them as needed.

We now show a mapping between the components of the causal models used in Def. 11 and the planning models defined in Def. 1. The exogenous variables $\mathcal{U}$ map to the static predicates (Helmert, 2009) in the domain, i.e., the ones that do not appear in the effect of any action; $\mathcal{V}$ maps to the non-static predicates; $\mathcal{R}$ maps each predicate to $\top$ if the predicate is true in a state, or $\bot$ when the predicate is false in a state; $\mathcal{F}$ calculates the value of each variable depending on the other variables that cause it. This is captured by the values of state predicates and executability variables being changed due to other state variables and decision variables.

5.2.3 Causal Soundness and Completeness

Before we prove that the models learned by Alg. 1 are causally correct, we first define the notions of causal soundness and completeness of a pair of models wrt. each other.

Definition 17. *Let $\vec{\mathcal{U}}$ and $\vec{\mathcal{V}}$ be the vectors of exogenous and endogenous variables, respectively; and Φ be the set of all boolean causal formulas expressible over variables in $\mathcal{V}$.*

A causal model M_1 is **causally complete** *with respect to another causal model M_2 if for all possible settings of exogenous variables, the causal relationships that are implied by the model M_1 are a superset of the set of causal relationships implied by the model M_2, i.e.,* $\forall \vec{u} \in \vec{\mathcal{U}}, \forall \vec{X}, \vec{X'} \subseteq \vec{\mathcal{V}}, \forall \varphi, \varphi' \in \Phi, \exists \vec{x} \in \vec{X}, \exists \vec{x'} \in \vec{X'}$ *s.t.* $\{\langle \vec{X}, \vec{u}, \varphi, \vec{x} \rangle : (\vec{X} = \vec{x}) \overset{(M_2,\vec{u})}{\rightsquigarrow} \varphi\} \subseteq \{\langle \vec{X'}, \vec{u}, \varphi', \vec{x'} \rangle : (\vec{X'} = \vec{x'}) \overset{(M_1,\vec{u})}{\rightsquigarrow} \varphi'\}$.

A causal model M_1 is **causally sound** *with respect to another causal model M_2 if for all possible settings of exogenous variables, the causal relationships implied by M_1 are a subset of the causal relationships implied by M_2, i.e.,* $\forall \vec{u} \in \vec{\mathcal{U}}, \forall \vec{X}, \vec{X'} \subseteq \vec{\mathcal{V}}, \forall \varphi, \varphi' \in \Phi, \exists \vec{x} \in \vec{X}, \exists \vec{x'} \in \vec{X'}$ *s.t.* $\{\langle \vec{X}, \vec{u}, \varphi, \vec{x} \rangle : (\vec{X} = \vec{x}) \overset{(M_1,\vec{u})}{\rightsquigarrow} \varphi\} \subseteq \{\langle \vec{X'}, \vec{u}, \varphi', \vec{x'} \rangle : (\vec{X'} = \vec{x'}) \overset{(M_2,\vec{u})}{\rightsquigarrow} \varphi'\}$.

We now show that the model(s) learned by AIA are causally sound and complete.

Theorem 4. *Given an agent $\mathcal{A}$ with a ground truth model $M^{\mathcal{A}}$ (unknown to the agent interrogation algorithm AIA), the action model M learned by AIA is causally sound and complete with respect to $M^{\mathcal{A}}$.*

Proof. We first show that M is sound with respect to $M^{\mathcal{A}}$. Assume that some $\vec{X} = \vec{x}$ is an actual cause of φ according to M in the setting $\vec{u}$, i.e., $(\vec{X} = \vec{x}) \overset{(M,\vec{u})}{\rightsquigarrow} \varphi$. Now by Thm 3, M contains palm tuples that are consistent with $M^{\mathcal{A}}$. Hence any palm tuple that is present in M will also be present in $M^{\mathcal{A}}$, implying that under the same setting $\vec{u}$ according to $M^{\mathcal{A}}$ $\vec{X} = \vec{x}$ is an actual cause of φ.

Now lets assume that some $\vec{X} = \vec{x}$ is an actual cause of φ according to $M^{\mathcal{A}}$ in the setting $\vec{u}$, i.e., $(\vec{X} = \vec{x}) \overset{(M^{\mathcal{A}},\vec{u})}{\rightsquigarrow} \varphi$. Now by Thm. 3, M contains exactly the same palm tuples as $M^{\mathcal{A}}$. Hence any palm tuple that is present in $M^a g$ will also be present in M, implying that under the same setting $\vec{u}$ according to M $\vec{X} = \vec{x}$ is an actual cause of φ. Hence the action model M learned by the agent interrogation algorithm are sound and complete with respect to the model $M^{\mathcal{A}}$. □

5.3 Complexity Analysis

Theoretically, the asymptotic complexity of AIA is $O(|P|^*| \times |A|)$, but it does not take into account how much computation is needed to answer the queries or to evaluate their responses. This complexity just shows the amount of computation needed in the worst case to derive the agent model by AIA. Here, we present a more detailed analysis of the complexity of AIA's queries using the results of relational query complexity by Vardi (1982).

This analysis takes into account the computational effort that the agent will have to put in to answer the queries. This is because we want to have minimal requirements on the agent to support AAM, and hence we don't want to ask questions that are too complex to answer. This analysis will also form a foundation for our future work on comparing different types of queries.

According to the notion of query complexity in Vardi (1982), a specific query is fixed in the language, then *data complexity* – given as function of size of databases – is found by applying this query to arbitrary databases. In the second notion of query complexity, if a specific database is fixed, then the *expression complexity* – given as a function of the length of expressions – is found by studying the complexity of applying queries represented by arbitrary expressions in the language. Finally, *combined complexity* – given as a function of combined size of the expressions and the database – is found by applying arbitrary queries in the language to arbitrary databases.

Theorem 5. *The membership classes of data, expression, and combined complexities of plan outcome queries are AC^0, ALOGTIME, and PTIME respectively.*

Proof. To analyze q_{PO}'s complexity, let us assume that the agent has stored the possible transitions it can make (in propositional form) using the relations $R(valid, s, a, s', succ)$, where $valid, succ \in \{\top, \bot\}$, $s, s' \in S$, $a \in A$; and $N(valid, n, n_+)$, where $valid \in \{\top, \bot\}$, $n, n_+ \in \mathbb{N}$, $0 \leq n \leq L$, and $0 \leq n_+ \leq L + 1$, where L is the maximum possible length of a plan in the q_{PO} queries. L can be an arbitrarily large number, and it does not matter as long as it is finite. Here, S and A are sets of grounded states and actions respectively. $succ$ is $\top$ if the action was executed successfully, and is $\bot$ if the action failed. $valid$ is $\top$ when none of the previous actions had $succ = \bot$. This stops an action to change a state if any of the previous actions failed, thereby preserving the state that resulted from a failed action. Whenever $succ = \bot$ or $valid = \bot$, $s = s'$ and $n = n_+$ signifying that applying an action where it is not applicable does not change the state.

Assuming the length of the query plan, $len(\pi) = D$, we can write a query in first order logic, equivalent to the plan outcome query as

$$\{(s_D, n_D) \mid \exists s_1, \ldots, \exists s_{D-1}, \exists succ_1, \ldots, \exists succ_{D-1}, \exists n_1, \ldots, \exists n_{D-1}$$
$$R(\top, s_0, a_1, s_1, succ_1) \wedge R(succ_1, s_1, a_2, s_2, succ_2) \wedge \cdots \wedge R(succ_{D-1}, s_{D-1}, a_D, s_D, \top) \wedge$$
$$N(\top, 0, n_1) \wedge N(succ_1, n_1, n_2) \wedge \cdots \wedge N(succ_{D-1}, n_{D-1}, n_D)\}$$

The output of the query contains the free variables $s_D = s_\ell$ and $n_D = \ell$. Such first order (FO) queries have the expression complexity and the combined complexity in PSPACE (Vardi, 1982). The data complexity class of FO queries is AC^0 (Immerman, 1987).

The following results use the analysis in Vardi (1995). The query analysis given above depends on how succinctly we can express the queries. In the FO query shown above, we have a lot of spurious quantified variables. We can reduce its complexity by using *bounded-variable* queries. Normally, queries in a language $\mathcal{L}$ assume an infinite supply $x_1, x_2, \ldots$ of individual variables. A *bounded-variable* version $\mathcal{L}^k$ of the language $\mathcal{L}$ is one which can be obtained by restricting the individual variables to be among $x_1, \ldots, x_k$, for $k > 0$. Using this, we can reduce the quantified variables in FO query shown earlier, and rewrite it more succinctly as an FO^k query by storing temporary query outputs.

$$E(succ, s, a, s', succ', n, n') = R(succ, s, a, s', succ') \wedge N(succ, n, n')$$
$$\alpha_1(succ, s, a_1, s', succ', n, n') = E(\top, s_0, a_1, s', succ', 0, n')$$

We then write subsequent queries corresponding to each step of the query plan as

$$\alpha_{i+1}(succ, s, a_{i+1}, s', succ', n, n') =$$
$$\exists s_1, \exists succ_1, \exists n_1 \{E(succ, s, a_{i+1}, s_1, succ_1, n_1) \wedge$$
$$\exists s, \exists succ, \exists n [succ = succ_1 \wedge s = s_1 \wedge$$
$$n = n_1 \wedge \alpha_i(succ, s, a_i, s', succ', n, n')]\}$$

Here i varies from 1 to D, and the value of k is 6 because of 6 quantified variables – $s, s_1, succ, succ_1, n$, and n_1. This reduces the expression and combined complexity of these queries to ALOGTIME and PTIME respectively. Note that these are the membership classes as it might be possible to write the queries more succinctly. □

6. Empirical Evaluation

We implemented AIA in Python to evaluate the efficacy of our approach.[4] In this implementation, initial states ($\mathcal{S}$, line 1 in Algorithm 1) were collected by making the agent perform random walks in a simulated environment. We used a maximum of 60 such random initial states for each domain in our experiments. The implementation assumes that the domains do not have any constants and that actions and predicates do not use repeated

4. Code available at `https://git.io/Jtpej`

Domain	$\|P^*\|$	$\|A\|$	$\|\hat{\mathbf{q}}\|$	$\mathbf{t}_\mu$ (ms)	$\mathbf{t}_\sigma$ (μs)
Gripper	5	3	17	18.0	0.2
Blocksworld	9	4	48	8.4	36
Miconic	10	4	39	9.2	1.4
Parking	18	4	63	16.5	806
Logistics	18	6	68	24.4	1.73
Satellite	17	5	41	11.6	0.87
Termes	22	7	134	17.0	110.2
Rovers	82	9	370	5.1	60.3
Barman	83	17	357	18.5	1605
Freecell	100	10	535	$2.24^\dagger$	$33.4^\dagger$

Table 3: The number of queries ($|\hat{q}|$), average time per query (t_μ), and variance of time per query (t_σ) generated by AIA with FD. Average and variance are calculated for 10 runs of AIA, each on a separate problem. †Time in sec.

variables (e.g., $at(?v, ?v)$), although these assumptions can be removed in practice without affecting the correctness of algorithms. The implementation is optimized to store the agent's answers to queries; hence the stored responses are used if a query is repeated. We evaluated three hypotheses using the experiments:

Hypothesis 1: The number of queries grows as we increase the number of *pal* tuples in the domains.

Hypothesis 2: The number of queries is lower than the observational learners to learn the complete model.

Hypothesis 3: The approach always learns the correct set of equivalent models.

We tested AIA on two types of agents: symbolic agents that use models from the IPC (unknown to AIA), and simulator agents that report states as images using PDDLGym. We wrote image classifiers for each predicate for the latter series of experiments and used them to derive state representations for use in the AIA algorithm. All experiments were executed on 5.0 GHz Intel i9-9900 CPUs with 64 GB RAM running Ubuntu 18.04.

The analysis presented below shows that AIA learns the correct model with a reasonable number of queries, and compares our results with the closest related work, FAMA (Aineto, Celorrio, & Onaindia, 2019). We use the metric of *model accuracy* in the following analysis: the number of correctly learned palm tuples normalized with the total number of palm tuples in $M^{\mathcal{A}}$.

6.1 Experiments with Symbolic Agents

We initialized the agent with one of the 10 IPC domain models, and ran AIA on the resulting agent. 10 different problem instances were used to obtain average performance estimates.

Table 3 shows that the number of queries required increases with the number of predicates and actions in the domain, hence proving Hypothesis 1. We used Fast Downward (Helmert, 2006) with LM-Cut heuristic (Helmert & Domshlak, 2009) to solve the planning problems. Since our approach is planner-independent, we also tried using FF (Hoffmann & Nebel, 2001) and the results were similar. The low variance shows that the method is stable across multiple runs.

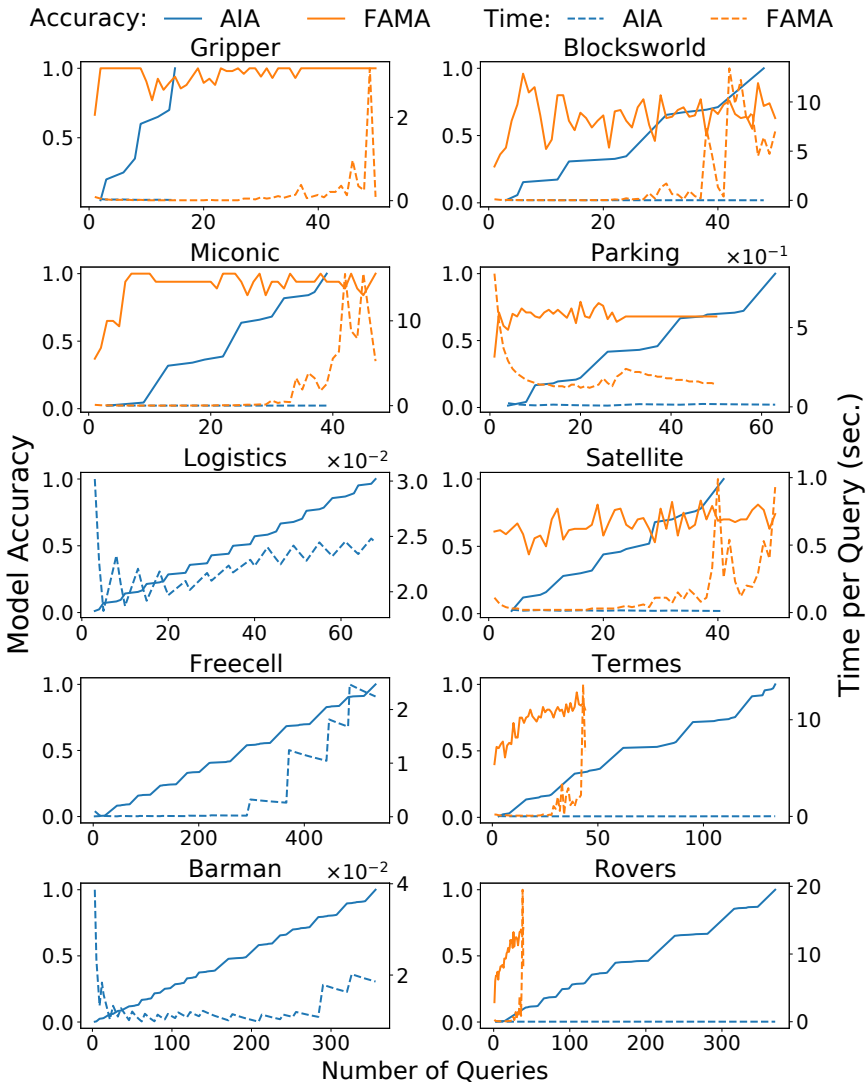

Figure 4: Performance comparison of AIA and FAMA in terms of model accuracy and time taken per query with an increasing number of queries.

6.2 Comparison with Observational Learner

We compare the performance of AIA with that of FAMA, state of the art observational learner, in terms of stability of the models learned and the time taken per query. Since the focus of our approach is on automatically generating useful traces, we provided FAMA randomly generated traces of length 3 (the length of the longest plans in AIA-generated queries) of the form used throughout this paper ($\langle s_I, a_1, a_2, a_3, s_G \rangle$).

Fig. 4 summarizes our findings. AIA takes lesser time per query and shows better convergence to the correct model, hence proving Hypothesis 2. FAMA sometimes reaches nearly accurate models faster, but its accuracy continues to oscillate, making it difficult to ascertain when the learning process should be stopped (we increased the number of traces provided to FAMA until it ran out of memory). This is because the solution to FAMA's

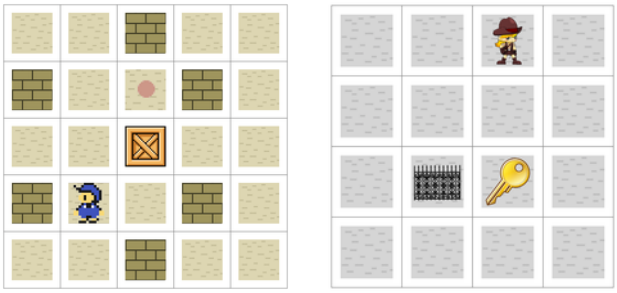

Figure 5: PDDLGym's simulated Sokoban (left) and Doors (right) environments used for the experiments.

internal planning problem introduces spurious palm tuples in its model if the input traces do not capture the complete domain dynamics. For Logistics, FAMA generated an incorrect planning problem, whereas for Freecell and Barman it ran out of memory (AIA also took considerable time for Freecell). Also, in domains with negative preconditions like Termes, FAMA was unable to learn the correct model. We used Madagascar (Rintanen, 2014) with FAMA as it is the preferred planner for it. We also tried FD and FF with FAMA, but as the original authors noted, it could not scale and ran out of memory on all but a few Blocksworld and Gripper problems where it was much slower than with Madagascar.

Also, not that AIA is able to learn the correct model for all the instances, hence proving Hypothesis 3.

6.3 Experiments with simulator agents

AIA can also be used with simulator agents that do not know about predicates and report states as images. To test this, we wrote classifiers for detecting predicates from images of simulator states in the PDDLGym (Silver & Chitnis, 2020) framework.

The classifiers are based on detecting objects in an image using colors (Duffy, Crowley, & Lacey, 2000; Khan et al., 2012).

This framework provides ground-truth PDDL models, thereby simplifying the estimation of accuracy. We initialized the agent with one of the two PDDLGym environments, Sokoban and Doors shown in Fig. 5. AIA inferred the correct model in both cases and the number of instantiated predicates, actions, and the average number of queries (over 5 runs) used to predict the correct model for Sokoban were 35, 3, and 201, and that for Doors were 10, 2, and 252.

7. Related Work

A number of researchers have explored the problem of learning agent models from observations of its behavior (Gil, 1994; X. Wang, 1994; Benson, 1995; Wu et al., 2007; Yang et al., 2007; Cresswell et al., 2009; Zhuo & Kambhampati, 2013). Such action-model learning approaches have also found practical applications in robot navigation (Balac et al., 2000), web-service description learning (Walsh & Littman, 2008), player behavior modeling (Krishnan et al., 2020), etc. To the best of our knowledge, ours is the first approach to

address the problem of generating query strategies for inferring relational models of black-box agents. We now present a detailed comparison of our work with the related works.

7.1 Passive Observations based Learners

Amir and Chang (2008) use logical filtering (Amir & Russell, 2003) to learn partially observable action models from the observation traces. Shahaf and Amir (2007); Zettlemoyer et al. (2008) and Shirazi and Amir (2011) also use logical filtering to acquire action models. Camacho and McIlraith (2019) present an approach for learning highly expressive LTL models from an agent's observed state trajectories using an oracle with knowledge of the target LTL representation. This oracle can also generate counterexamples when the estimated model differs from the true model, but it is not clear how to acquire such an oracle. Roy, Fisman, and Neider (2020) learns the models in Property Specification Language (PSL) with very little overhead as compared to learning LTL formulas. All these approaches learn models at the propositional level.

Genetic programming-based techniques like EvoCK (Aler et al., 1998), L2Plan (Levine & Humphreys, 2003), and LOUGA (Kučera & Barták, 2018) learn the domain rules by searching through a set of rules using genetic programming. LOCM (Cresswell et al., 2009), LOCM2 (Cresswell & Gregory, 2011), etc. present a class of algorithms that use finite-state machines to create action models from observed plan traces. ARMS (Yang et al., 2007), AMAN (Zhuo & Kambhampati, 2013), etc. leverage MAX-SAT to learn action models with partial or noisy traces.

FAMA (Aineto et al., 2019) reduces model recognition to a planning problem and can work with partial action sequences and/or state traces as long as correct initial and goal states are provided. While both FAMA and some other approaches like LOUGA (Kučera & Barták, 2018) require a post-processing step to update the learned model's preconditions to include the intersection of all states where an action is applied, it is not clear that such a process would necessarily converge to the correct model. Stern and Juba (2017), Juba et al. (2021), etc. learn safe action models for various settings leveraging intermediate states in execution traces. Our experiments indicate that such approaches exhibit oscillating behavior in terms of model accuracy because some data traces can include spurious predicates, which leads to spurious preconditions being added to the model's actions.

Bonet and Geffner (2020) and Rodriguez et al. (2021) present approaches for learning relational models using a SAT-based method when the action schema, predicates, etc. are not available. These approaches take as input a predesigned correct and complete directed graph encoding the structure of the entire state space. The authors note that their approach is viable for problems with small state spaces.

Online Learning Xu and Laird (2010) and Lamanna et al. (2021) use online learning to learn an action model incrementally. The idea is to incorporate new observations to improve the action model. These approaches even though incremental, do not focus on acquiring directed observations that will help it learn faster, but rater work with already available observations.

In contrast to these directions of research, our approach directly queries the agent and is guaranteed to converge to the true model while presenting a running estimate of the accuracy of the derived model; hence, it can be used in settings where the agent's model

changes due to learning or a software update. In such a scenario, our algorithm can restart to query the system, while approaches that derive models from observed plan traces would require arbitrarily long data collection sessions to get sufficient uncorrelated data.

7.2 Non-Passive Observation based Learners

Unlike the approaches that learn the action models using passively collected observations, there are some approaches that try to generate observations that help them direct the learning with lesser observations in general.

Active Learning The field of active learning (Settles, 2012) addresses the related problem of selecting which data labels to acquire for learning single-step decision-making models using statistical information measures. IRALe (Rodrigues et al., 2011) is one method that learns lifted transition modules by exploring actions in states where its partially learned preconditions almost hold. However, the effective feature set in active learning is the set of all possible plans, which makes conventional methods for evaluating the information gain of possible feature labelings infeasible. In contrast, our approach uses a hierarchical abstraction to select queries to ask while inferring a multistep decision-making (planning) model.

RL based Model Learning: Incremental Learning Model (Ng & Petrick, 2019) uses reinforcement learning to learn a non-stationary model without using plan traces, and requires extensive training to learn the full model correctly. Chitnis et al. (2021) present an approach for learning probabilistic relational models where they use goal sampling as a heuristic for generating relevant data, while we reduce that problem to query synthesis using planning. Their approach is shown to work well for stochastic environments, but puts a much higher burden on the AI system for inferring its model. This is because the AI system has to generate a conjunctive goal formula while maximizing exploration, find a plan to reach that goal, and correct the model as it collects observations while executing the plan.

Automata Learning: There is a large body of work on the active learning of automata of various types, namely DFA (Angluin, 1987) NFA (Oncina & García, 1992; Dupont, 1996), Moore machine (Giantamidis & Tripakis, 2016; Moerman, 2018), Mealy machine (Shahbaz & Groz, 2009), etc. Angluin (1987) proposed the earliest approaches for actively learning DFAs using the L* algorithm, which leveraged membership queries and equivalence queries. A significant limitation of L* is that these machines use grounded states as inputs, limiting their application to small state spaces. There have been multiple optimizations, including replacing exhaustive observation tables with decision trees (Kearns et al., 1994); carefully choosing suffixes as columns instead of adding all the prefixes of the counterexamples in the observation table (Rivest & Schapire, 1993); reorganizing the decision trees by combining the previous two approaches (Isberner et al., 2014); and using parameterized states in register automata (Cassel et al., 2015). Even with these optimizations, the number of membership queries required to learn the automata is quadratic in the input size. In contrast, the number of equivalence queries required is linear in the size of the input (Isberner et al., 2014). The above-mentioned approaches use the minimal adequate teacher (MAT) framework proposed by Angluin (1987). However, this has several drawbacks, which we cover below.

The first drawback of the MAT framework is that it needs a teacher who knows the correct model or approximates it to answer the equivalence queries correctly. Angluin (1988) showed that (i) using only membership queries, it is not possible to infer the correct DFA using a polynomial number of membership queries if the number of states of the target DFA is unknown; and (ii) even if the target number of states are known, an exponential number of membership queries are required. In our case even if we treat the simulator as a teacher, we don't need equivalence queries to guarantee that the model we learn is correct.

The second drawback of the MAT framework is that it needs knowledge of the input and output alphabet for working with membership and equivalence queries. The final drawback of the MAT framework is that the learned automata have control states that might not be readily interpretable as they may not map to actual environment states but some property of the environment. Our assessment approach alleviates these concerns as the final model is easily interpretable as the preconditions and effects are defined in terms the user understands.

Causal Accuracy of Learned Models: There have been some recent approaches that learn causal dynamics of a sequential decision-making system (Madumal et al., 2020; Z. Wang et al., 2022, 2024; Nashed et al., 2023; Amitai et al., 2024), but they either lack the theoretical guarantees we provide, need extra information about the dependency graph, or are not as scalable as our approach as we increase the domain size.

8. Conclusions and Future Work

We presented a novel approach for efficiently learning the internal model of an autonomous agent in a STRIPS-like form through query answering. Our theoretical and empirical results showed that the approach works well for both symbolic and simulator agents.

Extending our predicate classifier to handle noisy state detection, similar to prevalent approaches using classifiers to detect symbolic states (Konidaris et al., 2014; Asai & Fukunaga, 2018) is a good direction for future work. Some other promising extensions include replacing query and response communication interfaces between the agent and AAM with a natural language similar to Lindsay et al. (2017), or learning other representations like Zhuo et al. (2014). Additionally, in the future, such an assessment system can be used to make AI systems compliant with Level II assistive AI (Srivastava, 2021) by integrating this work with interfaces like JEDAI (Shah et al., 2022), PDSim (De Pellegrin & Petrick, 2024), JEDAI-Ed (Dobhal et al., 2024), etc.

Acknowledgements

We thank Shashank Rao Marpally for help in implementing an older version of the simulator agents. This work was supported in part by the ONR grant N00014-23-1-2416.

References

Aineto, D., Celorrio, S. J., & Onaindia, E. (2019). Learning Action Models With Minimal Observability. *Artificial Intelligence*, *275*, 104–137.

Aler, R., Borrajo, D., & Isasi, P. (1998). Genetic Programming of Control Knowledge for Planning. In *Proc. AIPS.*

Amir, E., & Chang, A. (2008). Learning Partially Observable Deterministic Action Models. *Journal of Artificial Intelligence Research*, *33*, 349–402.

Amir, E., & Russell, S. (2003). Logical Filtering. In *Proc. IJCAI.*

Amitai, Y., Septon, Y., & Amir, O. (2024). Explaining reinforcement learning agents through counterfactual action outcomes. In *Proc. AAAI.*

Angluin, D. (1987). Learning Regular Sets from Queries and Counterexamples. *Information and Computation*, *75*(2), 87-106.

Angluin, D. (1988, apr). Queries and concept learning. *Machine Learning*, *2*(4), 319–342.

Asai, M., & Fukunaga, A. (2018). Classical Planning in Deep Latent Space: Bridging the Subsymbolic-Symbolic Boundary. In *Proc. AAAI.*

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

Balac, N., Gaines, D., & Fisher, D. (2000). Learning Action Models for Navigation in Noisy Environments. In *ICML Workshop on Machine Learning of Spatial Knowledge.*

Benson, S. (1995). Inductive Learning of Reactive Action Models. In *Proc. ICML.*

Blondel, G., Arias, M., & Gavaldà, R. (2017, 03). Identifiability and Transportability in Dynamic Causal Networks. *International Journal of Data Science and Analytics*, *3*(2), 131–147.

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

Camacho, A., & McIlraith, S. A. (2019). Learning Interpretable Models Expressed in Linear Temporal Logic. In *Proc. ICAPS.*

Cassel, S., Howar, F., Jonsson, B., Merten, M., & Steffen, B. (2015). A Succinct Canonical Register Automaton Model. *Journal of Logical and Algebraic Methods in Programming*, *84*(1), 54-66.

Chitnis, R., Silver, T., Tenenbaum, J., Kaelbling, L. P., & Lozano-Perez, T. (2021). GLIB: Efficient Exploration for Relational Model-Based Reinforcement Learning via Goal-Literal Babbling. In *Proc. AAAI.*

Cresswell, S., & Gregory, P. (2011). Generalised Domain Model Acquisition from Action Traces. In *Proc. ICAPS.*

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

De Pellegrin, E., & Petrick, R. P. A. (2024). Planning domain simulation: An interactive system for plan visualisation. In *Proc. ICAPS.*

Dobhal, D., Nagpal, J., Karia, R., Verma, P., Nayyar, R. K., Shah, N., & Srivastava, S. (2024). Using explainable AI and hierarchical planning for outreach with robots. *arXiv preprint arXiv:2404.00808*.

Duffy, N., Crowley, J., & Lacey, G. (2000). Object Detection using Colour. In *Proc. ICPR.*

Dupont, P. (1996). Incremental Regular Inference. In *Proc. Third International Colloquium on Grammar Inference.*

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

Fox, M., & Long, D. (2003). PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, *20*(1), 61–124.

Giantamidis, G., & Tripakis, S. (2016). Learning moore machines from input-output traces. In *International symposium on formal methods.*

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

Halpern, J. Y. (2015). A Modification of the Halpern-Pearl Definition of Causality. In *Proc. IJCAI.*

Halpern, J. Y. (2016). *Actual Causality.* The MIT Press.

Helmert, M. (2004). A Planning Heuristic Based on Causal Graph Analysis. In *Proc. ICAPS.*

Helmert, M. (2006). The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, *26*, 191–246.

Helmert, M. (2009). Concise Finite-domain Representations for PDDL Planning Tasks. *Artificial Intelligence*, *173*(5-6), 503–535.

Helmert, M., & Domshlak, C. (2009). Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *Proc. ICAPS.*

Helmert, M., Haslum, P., & Hoffmann, J. (2007). Flexible Abstraction Heuristics for Optimal Sequential Planning. In *Proc. ICAPS.*

Hoffmann, J., & Nebel, B. (2001). The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, *14*, 253-302.

Immerman, N. (1987). *Expressibility as a complexity measure: Results and directions* (Tech. Rep. No. YALEU/DCS/TR-538). Department of Computer Science, Yale University.

Isberner, M., Howar, F., & Steffen, B. (2014). The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning. In *Proceedings of the international conference on runtime verification.*

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

Kanazawa, K., & Dean, T. (1989). A Model for Projection and Action. In *Proc. IJCAI.*

Kearns, M. J., Vazirani, U. V., & Vazirani, U. (1994). *An Introduction to Computational Learning Theory.* MIT press.

Khan, F. S., Anwer, R. M., van de Weijer, J., Bagdanov, A. D., Vanrell, M., & Lopez, A. M. (2012). Color Attributes for Object Detection. In *Proc. CVPR.*

Kim, B., Shah, J. A., & Doshi-Velez, F. (2015). Mind the gap: A generative approach to interpretable feature selection and extraction. In *Proc. NeurIPS.*

Konidaris, G., Kaelbling, L. P., & Lozano-Perez, T. (2014). Constructing Symbolic Representations for High-Level Planning. In *Proc. AAAI.*

Krishnan, A., Williams, A., & Martens, C. (2020). Towards Action Model Learning for Player Modeling. In *Proc. AIIDE.*

Kučera, J., & Barták, R. (2018). LOUGA: Learning Planning Operators Using Genetic Algorithms. In *Knowledge Management and Acquisition for Intelligent Systems.*

Lage, I., & Doshi-Velez, F. (2020). Learning interpretable concept-based models with human feedback. In *ICML Workshop on Human Interpretability in Machine Learning.*

Lamanna, L., Saetti, A., Serafini, L., Gerevini, A., & Traverso, P. (2021). Online Learning of Action Models for PDDL Planning. In *Proc. IJCAI.*

Levine, J., & Humphreys, D. (2003). Learning Action Strategies for Planning Domains Using Genetic Programming. In *Applications of Evolutionary Computing.*

Lindsay, A., Read, J., Ferreira, J., Hayton, T., Porteous, J., & Gregory, P. (2017). Framer: Planning Models from Natural Language Action Descriptions. In *Proc. ICAPS.*

Madumal, P., Miller, T., Sonenberg, L., & Vetere, F. (2020). Explainable reinforcement learning through a causal lens. In *Proc. AAAI.*

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

Mao, J., Lozano-Pérez, T., Tenenbaum, J. B., & Kaelbing, L. P. (2022). PDSketch: Integrated domain programming, learning, and planning. In *Proc. neurips.*

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., ... Wilkins, D. (1998). *PDDL – The Planning Domain Definition Language* (Tech. Rep. No. CVC TR-98-003/DCS TR-1165). Yale Center for Computational Vision and Control.

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

Moerman, J. (2018). Learning Product Automata. In *Proc. 14th International Conference on Grammatical Inference.*

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

Nashed, S. B., Mahmud, S., Goldman, C. V., & Zilberstein, S. (2023). Causal explanations for sequential decision making under uncertainty. In *Proc. AAMAS.*

Nayyar, R. K., Verma, P., & Srivastava, S. (2022). Differential assessment of black-box AI agents. In *Proc. AAAI.*

Ng, J. H. A., & Petrick, R. P. A. (2019). Incremental Learning of Planning Actions in Model-Based Reinforcement Learning. In *Proc. IJCAI.*

Oncina, J., & García, P. (1992). Inferring Regular Languages in Polynomial Update Time. *Pattern Recognition and Image Analysis*, *1*, 49–61.

Pearl, J. (1995). Causal Diagrams for Empirical Research. *Biometrika*, *82*(4), 669–688.

Pearl, J. (2009). *Causality: Models, Reasoning, and Inference*. Cambridge University Press.

Rintanen, J. (2014). Madagascar: Scalable Planning with SAT. In *Proc. 8th International Planning Competition.*

Rivest, R., & Schapire, R. (1993). Inference of Finite Automata Using Homing Sequences. *Information and Computation*, *103*(2), 299-347.

Rodrigues, C., Gérard, P., Rouveirol, C., & Soldano, H. (2011). Active Learning of Relational Action Models. In *Proc. ILP.*

Rodriguez, I. D., Bonet, B., Romero, J., & Geffner, H. (2021). Learning First-Order Representations for Planning from Black Box States: New Results. In *Proc. KR.*

Roy, R., Fisman, D., & Neider, D. (2020). Learning Interpretable Models in the Property Specification Language. In *Proc. IJCAI.*

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

Schulze, K. G., Shelby, R. N., Treacy, D. J., Wintersgill, M. C., VanLehn, K., & Gertner, A. (2000). Andes: An active learning, intelligent tutoring system for Newtonian Physics. *Themes in Education*, *1*(2), 115–136.

Settles, B. (2012). *Active Learning*. Morgan & Claypool Publishers.

Shah, N., Verma, P., Angle, T., & Srivastava, S. (2022). JEDAI: A system for skill-aligned explainable robot planning. In *Proc. AAMAS.*

Shahaf, D., & Amir, E. (2007). Logical Circuit Filtering. In *Proc. IJCAI.*

Shahbaz, M., & Groz, R. (2009). Inferring Mealy Machines. In *Proc. 2nd World Congress on Formal Methods.*

Shirazi, A., & Amir, E. (2011). First-order Logical Filtering. *Artificial Intelligence*, *175*(1), 193-219.

Silver, T., & Chitnis, R. (2020). PDDLGym: Gym Environments from PDDL Problems. In *ICAPS Workshop on Bridging the Gap Between AI Planning and Reinforcement Learning (PRL).*

Srivastava, S. (2021). Unifying Principles and Metrics for Safe and Assistive AI. In *Proc. AAAI.*

Srivastava, S., Russell, S., & Pinto, A. (2016). Metaphysics of Planning Domain Descriptions. In *Proc. AAAI.*

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

Vardi, M. Y. (1982). The complexity of relational query languages (extended abstract). In *Proc. 14th Annual ACM Symposium on Theory of Computing.*

Vardi, M. Y. (1995). On the complexity of bounded-variable queries (extended abstract). In *Proc. 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database*

Systems.

Verma, P., Karia, R., & Srivastava, S. (2023). Autonomous capability assessment of sequential decision-making systems in stochastic settings. In *Proc. NeurIPS.*

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

Verma, P., Marpally, S. R., & Srivastava, S. (2022). Discovering user-interpretable capabilities of black-box planning agents. In *Proc. KR.*

Walsh, T. J., & Littman, M. L. (2008). Efficient Learning of Action Schemas and Web-Service Descriptions. In *Proc. AAAI.*

Wang, X. (1994). Learning Planning Operators by Observation and Practice. In *Proc. AIPS.*

Wang, Z., Wang, C., Xiao, X., Zhu, Y., & Stone, P. (2024). Building minimal and reusable causal state abstractions for reinforcement learning. In *Proc. AAAI.*

Wang, Z., Xiao, X., Xu, Z., Zhu, Y., & Stone, P. (2022). Causal dynamics learning for task-independent state abstraction. In *Proc. ICML.*

Wu, K., Yang, Q., & Jiang, Y. (2007, June). ARMS: An Automatic Knowledge Engineering Tool for Learning Action Models for AI Planning. *Knowledge Engineering Review*, *22*(2), 135–152.

Xu, J., & Laird, J. (2010). Instance-Based Online Learning of Deterministic Relational Action Models. In *Proc. AAAI.*

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

Zettlemoyer, L. S., Pasula, H. M., & Kaelbling, L. P. (2008). Logical Particle Filtering. In *Probabilistic, Logical and Relational Learning - A Further Synthesis.*

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

Zhuo, H. H., Muñoz-Avila, H., & Yang, Q. (2014). Learning Hierarchical Task Network Domains from Partially Observed Plan Traces. *Artificial Intelligence*, *212*, 134 - 157.