Differential Assessment of Black-Box AI Agents

Rashmeet Kaur Nayyar*, **Pulkit Verma***, and **Siddharth Srivastava**
Autonomous Agents and Intelligent Robots Lab,
School of Computing and Augmented Intelligence, Arizona State University, AZ, USA
{rmnayyar, verma.pulkit, siddharths}@asu.edu

Abstract

Much of the research on learning symbolic models of AI agents focuses on agents with stationary models. This assumption fails to hold in settings where the agent's capabilities may change as a result of learning, adaptation, or other post-deployment modifications. Efficient assessment of agents in such settings is critical for learning the true capabilities of an AI system and for ensuring its safe usage. In this work, we propose a novel approach to *differentially* assess black-box AI agents that have drifted from their previously known models. As a starting point, we consider the fully observable and deterministic setting. We leverage sparse observations of the drifted agent's current behavior and knowledge of its initial model to generate an active querying policy that selectively queries the agent and computes an updated model of its functionality. Empirical evaluation shows that our approach is much more efficient than re-learning the agent model from scratch. We also show that the cost of differential assessment using our method is proportional to the amount of drift in the agent's functionality.

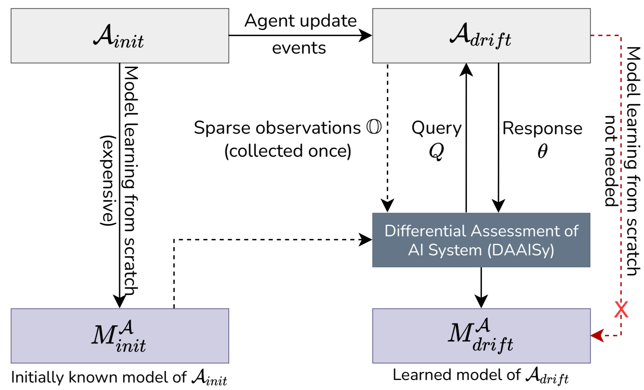

Figure 1: The Differential Assessment of AI System (DAAISy) takes as input the initially known model of the agent prior to model drift, available observations of the updated agent's behavior, and performs a selective dialog with the black-box AI agent to output its updated model through efficient model learning.

1 Introduction

With increasingly greater autonomy in AI systems in recent years, a major problem still persists and has largely been overlooked: how do we accurately predict the behavior of a black-box AI agent that is evolving and adapting to changes in the environment it is operating in? And how do we ensure its reliable and safe usage? Numerous factors could cause unpredictable changes in agent behaviors: sensors and actuators may fail due to physical damage, the agent may adapt to a dynamic environment, users may change deployment and use-case scenarios, etc. Most prior work on the topic presumes that the functionalities and the capabilities of AI agents are static, while some works start with a *tabula-rasa* and learn the entire model from scratch. However, in many real-world scenarios, the agent model is transient and only parts of its functionality change at a time.

Bryce, Benton, and Boldt (2016) address a related problem where the system learns the updated mental model of a user using particle filtering given prior knowledge about the user's mental model. However, they assume that the entity being modeled can tell the learning system about flaws in the learned model if needed. This assumption does not hold in settings where the entity being modeled is a black-box AI system: most such systems are either implemented using inscrutable representations or otherwise lack the ability to automatically generate a model of their functionality (what they can do and when) in terms the user can understand. The problem of efficiently assessing, in human-interpretable terms, the functionality of such a non-stationary AI system has received little research attention.

The primary contribution of this paper is an algorithm for *differential assessment* of black-box AI systems (Fig. 1). This algorithm utilizes an initially known interpretable model of the agent as it was in the past, and a small set of observations of agent execution. It uses these observations to develop an incremental querying strategy that avoids the full cost of assessment from scratch and outputs a revised model of the agent's new functionality. One of the challenges in learning agent models from observational data is that reductions in agent functionality often do not correspond to specific "evidence" in behavioral observations, as the agent may not visit states where certain useful actions are no longer ap-

*Equal contribution. Alphabetical order.

plicable. Our analysis shows that if the agent can be placed in an "optimal" planning mode, differential assessment can indeed be used to query the agent and recover information about reduction in functionality. This "optimal" planning mode is not necessarily needed for learning about increase in functionality. Empirical evaluations on a range of problems clearly demonstrate that our method is much more efficient than re-learning the agent's model from scratch. They also exhibit the desirable property that the computational cost of differential assessment is proportional to the amount of drift in the agent's functionality.

Running Example Consider a battery-powered rover with limited storage capacity that collects soil samples and takes pictures. Assume that its planning model is similar to IPC domain Rovers (Long and Fox 2003). It has an action that collects a rock sample at a waypoint and stores it in a storage iff it has at least half of the battery capacity remaining. Suppose there was an update to the rover's system and as a result of this update, the rover can now collect the rock sample only when its battery is full, as opposed to at least half-charged battery that it needed before. Mission planners familiar with the earlier system and unaware about the exact updates in the functionality of the rover would struggle to collect sufficient samples. This could jeopardise multiple missions if it is not detected in time.

This example illustrates how our system could be of value by differentially detecting such a drift in the functionality of a black-box AI system and deriving its true functionality.

The rest of this paper is organized as follows: The next section presents background terminology. This is followed by a formalization of the differential model assessment problem in Section 3. Section 4 presents our approach for differential assessment by first identifying aspects of the agent's functionality that may be affected (Section 4.1) followed by the process for selectively querying the agent using a primitive set of queries. We present empirical evaluation of the efficiency of our approach on randomly generated benchmark planning domains in Section 5. Finally, we discuss relevant related work in Section 6 and conclude in Section 7.

2 Preliminaries

We consider models that express an agent's functionalities in the form of STRIPS-like planning models (Fikes and Nilsson 1971; McDermott et al. 1998; Fox and Long 2003) as defined below.

Definition 1. A planning domain model is a tuple $M = \langle P, A \rangle$, where $P = \{p_1^{r_1}, \dots, p_n^{r_n}\}$ is a finite set of predicates with arities $r_i, i \in [1, n]$; and $A = \{a_1, \dots, a_k\}$ is a finite set of parameterized relational actions. Each action $a_i \in A$ is represented as a tuple $\langle header(a_i), \mathit{pre}(a_i), \mathit{eff}(a_i) \rangle$, where $header(a_i)$ represents the action header consisting of the name and parameters for the action a_i, $pre(a_i)$ represents the conjunction of positive or negative literals that must be true in a state where the action a_i is applicable, and $\mathit{eff}(a_i)$ is the conjunction of positive or negative literals that become true as a result of execution of the action a_i.

In the rest of the paper, we use the term "model" to refer to planning domain models and use closed-world assumption as used in the Planning Domain Definition Language (PDDL) (McDermott et al. 1998). 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 context. An action $a \in A$ is applicable in a state $s \in S$ if $s \models \mathit{pre}(a)$. The result of executing a 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.

A literal corresponding to a predicate $p \in P$ can appear in $\mathit{pre}(a)$ or $\mathit{eff}(a)$ of an action $a \in A$ if and only if it can be instantiated using a subset of parameters of a. E.g., consider an action *navigate (?rover ?src ?dest)* and a predicate *(can_traverse ?rover ?x ?y)* in the Rovers domain discussed earlier. Suppose a literal corresponding to predicate *(can_traverse ?rover ?x ?y)* can appear in the precondition and/or the effect of *navigate (?rover ?src ?dest)* action. Assuming we know *?x* and *?y* in *can_traverse*, and *?src* and *?dest* in *navigate* are of the same type *waypoint*, the possible lifted instantiations of predicate *can_traverse* compatible with action *navigate* are *(can_traverse ?rover ?src ?dest)*, *(can_traverse ?rover ?dest ?src)*, *(can_traverse ?rover ?src ?src)*, and *(can_traverse ?rover ?dest ?dest)*. The number of parameters in a predicate $p \in P$ that is relevant to an action $a \in A$, i.e., instantiated using a subset of parameters of the action a, is bounded by the maximum arity of the action a. We formalize this notion of lifted instantiations of a predicate with an action as follows:

Definition 2. Given a finite set of predicates $P = \{p_1^{r_1}, \dots, p_n^{r_n}\}$ with arities r_i, $i \in [1, n]$; and a finite set of parameterized relational actions $A = \{a_1^{\psi_1}, \dots, a_k^{\psi_k}\}$ with arities ψ_j and parameters $\mathit{par}(a_j^{\psi_j}) = \langle \alpha_1, \dots, \alpha_{\psi_j} \rangle$, $j \in [1, k]$, the set of *lifted instantiations of predicates* P^* is defined as the collection $\{p_i(\sigma(x_1), \dots, \sigma(x_{r_i})) \,|\, p_i \in P, a \in A, \sigma : \{x_1, \dots, x_{r_i}\} \rightarrow \mathit{par}(a)\}$.

2.1 Representing Models

We represent a model M using the set of all possible *pal-tuples* Γ_M of the form $\gamma = \langle p, a, \ell \rangle$, where a is a parameterized action header for an action in A, $p \in P^*$ is a possible lifted instantiation of a predicate in P, and $\ell \in \{\mathit{pre}, \mathit{eff}\}$ denotes a location in a, precondition or effect, where p can appear. A model M is thus a function $\mu_M : \Gamma_M \rightarrow \{+, -, \emptyset\}$ that maps each element in Γ_M to a *mode* in the set $\{+, -, \emptyset\}$. The assigned mode for a *pal-tuple* $\gamma \in \Gamma_M$ denotes whether p is present as a positive literal ($+$), as a negative literal ($-$), or absent ($\emptyset$) in the precondition ($\ell = \mathit{pre}$) or effect ($\ell = \mathit{eff}$) of the action header a.

This formulation of models as *pal-tuples* allows us to view the modes for any predicate in an action's precondition and effect independently. However, at times it is useful to consider a model at a granularity of relationship between a predicate and an action. We address this by representing a model M as a set of *pa-tuples* Λ_M of the form $\langle p, a \rangle$ where a is a parameterized action header for an action in A, and $p \in P^*$ is a possible lifted instantiation of a predicate in P. Each *pa-tuple* can take a value of the form $\langle m_{\mathit{pre}}, m_{\mathit{eff}} \rangle$, where m_{pre} and m_{eff} represents the mode in which p appears in the precondition and effect of a, respectively. Since a predicate cannot appear as a positive (or negative) literal in both the precondition and effect of an action, $\langle +,+\rangle$ and $\langle -,-\rangle$ are not in the range of values that *pa-tuples* can take. Henceforth, in the context of a *pal-tuple* or a *pa-tuple*, we refer to a as an action instead of an action header.

Measure of model difference Given two models $M_1 = \langle P, A_1\rangle$ and $M_2 = \langle P, A_2\rangle$, defined over the same sets of predicates P and action headers A, the difference between the two models $\Delta(M_1, M_2)$ is defined as the number of *pal-tuples* that differ in their modes in M_1 and M_2, i.e., $\Delta(M_1, M_2) = |\{\gamma \in P \times A \times \{+,-,\emptyset\} | \mu_{M_1}(\gamma) \neq \mu_{M_2}(\gamma)\}|$.

2.2 Abstracting Models

Several authors have explored the use of abstraction in planning (Sacerdoti 1974; Giunchiglia and Walsh 1992; Helmert, Haslum, and Hoffmann 2007; Bäckström and Jonsson 2013; Srivastava, Russell, and Pinto 2016). We define an abstract model as a model that does not have a mode assigned for at least one of the *pal-tuples*. Let Γ_M be the set of all possible *pal-tuples*, and ? be an additional possible value that a *pal-tuple* can take. Assigning ? mode to a *pal-tuple* denotes that its mode is unknown. An abstract model M is thus a function $\mu_M : \Gamma_M \rightarrow \{+,-,\emptyset,?\}$ that maps each element in Γ_M to a *mode* in the set $\{+,-,\emptyset,?\}$. Let $\mathcal{U}$ be the set of all abstract and concrete models that can possibly be expressed by assigning modes in $\{+,-,\emptyset,?\}$ to each *pal-tuple* $\gamma \in \Gamma_M$. We now formally define model abstraction as follows:

Definition 3. Given models M_1 and M_2, M_2 is an abstraction of M_1 over the set of all possible *pal-tuples* Γ iff $\exists \Gamma_2 \subseteq \Gamma$ s.t. $\forall \gamma \in \Gamma_2$, $\mu_{M_2}(\gamma) = ?$ and $\forall \gamma \in \Gamma \setminus \Gamma_2$, $\mu_{M_2}(\gamma) = \mu_{M_1}(\gamma)$.

2.3 Agent Observation Traces

We assume limited access to a set of observation traces $\mathbb{O}$, collected from the agent, as defined below.

Definition 4. An *observation trace* o is a 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 $\forall i \in [1, n]\ \ a_i(s_{i-1}) = s_i$.

These observation traces can be split into multiple action triplets as defined below.

Definition 5. Given an observation trace $o = \langle s_0, a_1, s_1, a_2, \ldots, s_{n-1}, a_n, s_n\rangle$, an *action triplet* is a 3-tuple sub-sequence of o of the form $\langle s_{i-1}, a_i, s_i\rangle$, where $i \in [1, n]$ and applying an action a_i in state s_{i-1} results in state s_i, i.e., $a_i(s_{i-1}) = s_i$. The states s_{i-1} and s_i are called pre- and post-states of action a_i, respectively.

An action triplet $\langle s_{i-1}, a_i, s_i\rangle$ is said to be *optimal* if there does not exist an action sequence (of length ≥ 1) that takes the agent from state s_{i-1} to s_i with total action cost less than that of action a_i, where each action a_i has unit cost.

2.4 Queries

We use queries to actively gain information about the functionality of an agent to learn its updated model. We assume that the agent can respond to a query using a simulator. The availability of such agents with simulators is a common assumption as most AI systems already use simulators for design, testing, and verification.

We use a notion of queries similar to Verma, Marpally, and Srivastava (2021), to perform a dialog with an autonomous agent. These queries use an agent to determine what happens if it executes a sequence of actions in a given initial state. E.g., in the rovers domain, the rover could be asked: what happens when the action *sample_rock(rover1 storage1 waypoint1)* is executed in an initial state $\{$*(equipped_rock_analysis rover1), (battery_half rover1), (at rover1 waypoint1)*$\}$?

Formally, a *query* is a function that maps an agent to a response, which we define as:

Definition 6. Given a set of predicates P, a set of actions A, and a set of objects O, a *query* $Q\langle s, \pi\rangle : \mathcal{A} \rightarrow \mathbb{N} \times S$ is parameterized by a start state $s_I \in S$ and a plan $\pi = \langle a_1, \ldots, a_N\rangle$, where S is the state space over P and O, and $\{a_1, \ldots, a_N\}$ is a subset of action space over A and O. It maps agents to responses $\theta = \langle n_F, s_F\rangle$ such that n_F is the length of the longest prefix of π that $\mathcal{A}$ can successfully execute and $s_F \in S$ is the result of that execution.

Responses to such queries can be used to gain useful information about the model drift. E.g., consider an agent with an internal model $M^{\mathcal{A}}_{drift}$ as shown in Tab. 1. If a query is posed asking what happens when the action *sample_rock(rover1 storage1 waypoint1)* is executed in an initial state $\{$*(equipped_rock_analysis rover1), (battery_half rover1), (at rover1 waypoint1)*$\}$, the agent would respond $\langle 0, \{$*(equipped_rock_analysis rover1), (battery_half rover1), (at rover1 waypoint1)*$\}\rangle$, representing that it was not able to execute the plan, and the resulting state was $\{$*(equipped_rock_analysis rover1), (battery_half rover1), (at rover1 waypoint1)*$\}$ (same as the initial state in this case). Note that this response is inconsistent with the model $M^{\mathcal{A}}_{init}$, and it can help in identifying that the precondition of action *sample_rock(?r ?s ?w)* has changed.

3 Formal Framework

Our objective is to address the problem of differential assessment of black-box AI agents whose functionality may have changed from the last known model. Without loss of generality, we consider situations where the set of action headers is same because the problem of differential assessment with changing action headers can be reduced to that with uniform action headers. This is because if the set of actions has increased, new actions can be added with empty preconditions and effects to $M^{\mathcal{A}}_{init}$, and if it has decreased, $M^{\mathcal{A}}_{init}$ can be reduced similarly. We assume that the predicate vocabulary used in the two models is the same; extension to situations where the vocabulary changes can be used to model open-world scenarios. However, that extension is beyond the scope of this paper.

Suppose an agent $\mathcal{A}$'s functionality was known as a model $M^{\mathcal{A}}_{init} = \langle P, \mathcal{A}_{init}\rangle$, and we wish to assess its current functionality as the model $M^{\mathcal{A}}_{drift} = \langle P, \mathcal{A}_{drift}\rangle$. The drift in the functionality of the agent can be measured by changes in the

Model	Precondition	Effect
$M^{\mathcal{A}}_{init}$	*(equipped_rock_analysis?r)* *(battery_half ?r)* *(at ?r ?w)*	*(rock_sample_taken?r)* *(store_full ?r ?s)* $\neg$*(battery_half ?r)* *(battery_reserve ?r)*
$M^{\mathcal{A}}_{drift}$	*(equipped_rock_analysis?r)* *(battery_full ?r)* *(at ?r ?w)*	*(rock_sample_taken ?r)* *(store_full ?r ?s)* $\neg$*(battery_full ?r)* *(battery_half ?r)*

Table 1: *sample_rock (?r ?s ?w)* action of the agent $\mathcal{A}$ in $M^{\mathcal{A}}_{init}$ and a possible drifted model $M^{\mathcal{A}}_{drift}$.

preconditions and/or effects of all the actions in $\mathcal{A}_{init}$. The extent of the drift between $M^{\mathcal{A}}_{init}$ and $M^{\mathcal{A}}_{drift}$ is represented as the model difference $\Delta(M^{\mathcal{A}}_{init}, M^{\mathcal{A}}_{drift})$.

We formally define the problem of differential assessment of an AI agent below.

Definition 7. Given an agent $\mathcal{A}$ with a functionality model $M^{\mathcal{A}}_{init}$, and a set of observations $\mathbb{O}$ collected using its current version of $\mathcal{A}_{drift}$ with unknown functionality $M^{\mathcal{A}}_{drift}$, the *differential model assessment* problem $\langle M^{\mathcal{A}}_{init}, M^{\mathcal{A}}_{drift}, \mathbb{O}, \mathcal{A}\rangle$ is defined as the problem of inferring $\mathcal{A}$ in form of $M^{\mathcal{A}}_{drift}$ using the inputs $M^{\mathcal{A}}_{init}$, $\mathbb{O}$, and $\mathcal{A}$.

We wish to develop solutions to the problem of differential assessment of AI agents that are more efficient than re-assessment from scratch.

3.1 Correctness of Assessed Model

We now discuss the properties that a model, which is a solution to the differential model assessment problem, should satisfy. A critical property of such models is that they should be consistent with the observation traces. We formally define consistency of a model w.r.t. an observation trace as follows:

Definition 8. Let o be an observation trace $\langle s_0, a_1, s_1, a_2, \ldots, s_{n-1}, a_n, s_n\rangle$. A model $M = \langle P, A\rangle$ is *consistent with the observation trace* o iff $\forall i \in \{1,..,n\}\ \exists a \in A$ and a_i is a grounding of action a *s.t.* $s_{i-1} \models pre(a_i) \wedge \forall\, l \in \mathit{eff}(a_i)\ s_i \models l$.

In addition to being consistent with observation traces, a model should also be consistent with the queries that are asked and the responses that are received while actively inferring the model of the agent's new functionality. We formally define consistency of a model with respect to a query and a response as:

Definition 9. Let $M = \langle P, A\rangle$ be a model; O be a set of objects; $Q = \langle s_I, \pi = \langle a_1, \ldots a_n\rangle\rangle$ be a query defined using P, A, and O, and let $\theta = \langle n_F, s_F\rangle$, $(n_F \leq n)$ be a response to Q. M is *consistent with the query-response* $\langle Q, \theta\rangle$ iff there exists an observation trace $\langle s_I, a_1, s_1, \ldots, a_{n_F}, s_{n_F}\rangle$ that M is consistent with and $s_{n_F} \not\models pre(a_{n_F+1})$ where $pre(a_{n_F+1})$ is the precondition of a_{n_F+1} in M.

We now discuss our methodology for solving the problem of differential assessment of AI systems.

4 Differential Assessment of AI Systems

Differential **A**ssessment of **AI Sy**stems (Alg. 1) -- DAAISy -- takes as input an agent $\mathcal{A}$ whose functionality has drifted, the model $M^{\mathcal{A}}_{init} = \langle P, A\rangle$ representing the previously known functionality of $\mathcal{A}$, a set of arbitrary observation traces $\mathbb{O}$, and a set of random states $\mathcal{S} \subseteq S$. Alg. 1 returns a set of updated models $\mathcal{M}^{\mathcal{A}}_{drift}$, where each model $M^{\mathcal{A}}_{drift} \in \mathcal{M}^{\mathcal{A}}_{drift}$ represents $\mathcal{A}$'s updated functionality and is consistent with all observation traces $o \in \mathbb{O}$.

A major contribution of this work is to introduce an approach to make inferences about not just the expanded functionality of an agent but also its reduced functionality using a limited set of observation traces. Situations where the scope of applicability of an action reduces, i.e., the agent can no longer use an action a to reach state s' from state s while it could before (e.g., due to addition of a precondition literal), are particularly difficult to identify because observing its behavior does not readily reveal what it cannot do in a given state. Most observation based action-model learners, even when given access to an incomplete model to start with, fail to make inferences about reduced functionality. DAAISy uses two principles to identify such a functionality reduction. First, it uses active querying so that the agent can be made to reveal failure of reachability, and second, we show that if the agent can be placed in optimal planning mode, plan length differences can be used to infer a reduction in functionality.

DAAISy performs two major functions; it first identifies a salient set of *pal-tuples* whose modes were likely affected (line 1 of Alg. 1), and then infers the mode of such affected *pal-tuples* accurately through focused dialog with the agent (line 2 onwards of Alg. 1). In Sec. 4.1, we present our method for identifying a salient set of potentially affected *pal-tuples* that contribute towards expansion in the functionality of the agent through inference from available arbitrary observations. We then discuss the problem of identification of *pal-tuples* that contribute towards reduction in the functionality of the agent and argue that it cannot be performed using successful executions in observations of satisficing behavior. We show that *pal-tuples* corresponding to reduced functionality can be identified if observations of optimal behavior of the agent are available (Sec. 4.1). Finally, we present how we infer the nature of changes in all affected *pal-tuples* through a query-based interaction with the agent (Sec. 4.2) by building upon the Agent Interrogation Algorithm (AIA) (Verma, Marpally, and Srivastava 2021). Identifying affected *pal-tuples* helps reduce the computational cost of querying as opposed to the exhaustive querying strategy used by AIA. We now discuss the two major functions of Alg. 1 in detail.

4.1 Identifying Potentially Affected pal-tuples

We identify a reduced set of *pal-tuples* whose modes were potentially affected during the model drift, denoted by Γ_δ, using a small set of available observation traces $\mathbb{O}$. We draw two kinds of inferences from these observation traces: inferences about expanded functionality, and inferences about reduced functionality. We discuss our method for inferring

Algorithm 1: Differential Assessment of AI Systems

Input: $M^{\mathcal{A}}_{init}$, $\mathbb{O}$, $\mathcal{A}$, $\mathcal{S}$
Output: $\mathcal{M}^{\mathcal{A}}_{drift}$

$\Gamma_\delta \leftarrow$ *identify_affected_pals()*
$M_{abs} \leftarrow$ set *pal-tuples* in $M^{\mathcal{A}}_{init}$ corresponding to Γ_δ to ?
$\mathcal{M}^{\mathcal{A}}_{drift} \leftarrow \{M_{abs}\}$
for each γ in Γ_δ **do**
  for each M_{abs} in $\mathcal{M}^{\mathcal{A}}_{drift}$ **do**
    $\mathcal{M}_{abs} \leftarrow M_{abs} \times \{\gamma^+, \gamma^-, \gamma^\emptyset\}$
    $\mathcal{M}_{sieved} \leftarrow \{\}$
    if action corresponding to γ: γ_a in $\mathbb{O}$ **then**
      $s_{pre} \leftarrow$ *states_where_*γ_a*_applicable*$(\mathbb{O}, \gamma_a)$
      $Q \leftarrow \langle s_{pre} \setminus \{\gamma_p \cup \neg\gamma_p\}, \gamma_a \rangle$
      $\theta \leftarrow$ *ask_query*$(\mathcal{A}, Q)$
      $\mathcal{M}_{sieved} \leftarrow$ *sieve_models*$(\mathcal{M}_{abs}, Q, \theta)$
    else
      for each pair $\langle M_i, M_j \rangle$ in $\mathcal{M}_{abs}$ **do**
        $Q \leftarrow$ *generate_query*(M_i, M_j, γ, S)
        $\theta \leftarrow$ *ask_query*$(\mathcal{A}, Q)$
        $\mathcal{M}_{sieved} \leftarrow$ *sieve_models*$(\{M_i, M_j\}, Q, \theta)$
      end for
    end if
    $\mathcal{M}_{abs} \leftarrow \mathcal{M}_{abs} \setminus \mathcal{M}_{sieved}$
  end for
  $\mathcal{M}^{\mathcal{A}}_{drift} \leftarrow \mathcal{M}_{abs}$
end for

Γ_δ for both types of changes in the functionality below.

Expanded functionality To infer expanded functionality of the agent, we use the previously known model of the agent's functionality and identify its differences with the possible behaviors of the agent that are consistent with $\mathbb{O}$. To identify the *pal-tuples* that directly contribute to an expansion in the agent's functionality, we perform an analysis similar to Stern and Juba (2017), but instead of bounding the predicates that can appear in each action's precondition and effect, we bound the range of possible values that each *pa-tuple* in $M^{\mathcal{A}}_{drift}$ can take using Tab. 2. For any *pa-tuple*, a direct comparison between its value in $M^{\mathcal{A}}_{init}$ and possible inferred values in $M^{\mathcal{A}}_{drift}$ provides an indication of whether it was affected.

To identify possible values for a *pa-tuple* $\langle p, a \rangle$, we first collect a set of all the action-triplets from $\mathbb{O}$ that contain the action a. For a given predicate p and state s, if $s \models p$ then the presence of predicate p is represented as *pos*, similarly, if $s \models \neg p$ then the presence of predicate p is represented as *neg*. Using this representation, a tuple of predicate presence $\in$ {*(pos,pos)*, *(pos,neg)*, *(neg,pos)*, *(neg,neg)*} is determined for the *pa-tuple* $\langle p, a \rangle$ for each action triplet $\langle s, a, s' \rangle \in \mathbb{O}$ by analyzing the presence of predicate p in the pre- and post-states of the action triplets. Possible values of the *pa-tuple* that are consistent with $\mathbb{O}$ are directly inferred from the Tab. 2 using the inferred tuples of predicate presence. E.g., for a *pa-tuple*, the values $\langle +, - \rangle$ and $\langle \emptyset, - \rangle$ are consistent with (pos, neg), whereas, only $\langle \emptyset, + \rangle$ is consistent with

$\langle m_{pre}, m_{eff} \rangle$	*(pos,pos)*	*(pos,neg)*	*(neg,pos)*	*(neg,neg)*
$\langle +, - \rangle$	✗	✓	✗	✗
$\langle +, \emptyset \rangle$	✓	✗	✗	✗
$\langle -, + \rangle$	✗	✗	✓	✗
$\langle -, \emptyset \rangle$	✗	✗	✗	✓
$\langle \emptyset, + \rangle$	✓	✗	✓	✗
$\langle \emptyset, - \rangle$	✗	✓	✗	✓
$\langle \emptyset, \emptyset \rangle$	✓	✗	✗	✓

Table 2: Each row represents a possible value $\langle m_{pre}, m_{eff} \rangle$ for a *pa-tuple* $\langle p, a \rangle$. Each column represents a possible tuple representing presence of predicate p in the pre- and post-states of an action triplet $\langle s_i, a, s_{i+1} \rangle$ (discussed in Sec.4.1). The cells represent whether a value for *pa-tuple* is consistent with an action triplet in observation traces.

(pos, pos) and (neg, pos) tuples of predicate presence that are inferred from $\mathbb{O}$.

Once all the possible values for each *pa-tuple* in $M^{\mathcal{A}}_{drift}$ are inferred, we identify *pa-tuples* whose previously known value in $M^{\mathcal{A}}_{init}$ is no longer possible due to inconsistency with $\mathbb{O}$. The *pal-tuples* corresponding to such *pa-tuples* are added to the set of potentially affected *pal-tuples* Γ_δ. Our method also infers the correct modes of a subset of *pal-tuples*. E.g., consider a predicate p and two actions triplets in $\mathbb{O}$ of the form $\langle s_1, a, s'_1 \rangle$ and $\langle s_2, a, s'_2 \rangle$ that satisfy $s_1 \models p$ and $s_2 \models \neg p$. Such an observation clearly indicates that p is not in the precondition of action a, i.e., mode for $\langle p, a \rangle$ in the precondition is $\emptyset$. Such inferences of modes are used to update the known functionality of the agent. We remove such *pal-tuples*, whose modes are already inferred, from Γ_δ.

A shortcoming of direct inference from successful executions in available observation traces is that it cannot learn any reduction in the functionality of the agent, as discussed in the beginning of Sec. 4. We now discuss our method to address this limitation and identify a larger set of potentially affected *pal-tuples*.

Reduced functionality We conceptualize reduction in functionality as an increase in the optimal cost of going from one state to another. More precisely, reduction in functionality represents situations where there exist states s_i, s_j such that the minimum cost of going from s_i to s_j is higher in $M^{\mathcal{A}}_{drift}$ than in $M^{\mathcal{A}}_{init}$. In this paper, this cost refers to the number of steps between the pair of states as we consider unit action costs. This notion encompasses situations with reductions in reachability as a special case. In practice, a reduction in functionality may occur if the precondition of at least one action in $M^{\mathcal{A}}_{drift}$ has new *pal-tuples*, or the effect of at least one of its actions has new *pal-tuples* that conflict with other actions required for reaching certain states.

Our notion of reduced functionality captures all the variants of reduction in functionality. However, for clarity, we illustrate an example that focuses on situations where precondition of an action has increased. Consider the case from Tab. 1 where $\mathcal{A}$'s model gets updated from $M^{\mathcal{A}}_{init}$ to $M^{\mathcal{A}}_{drift}$.

The action *sample_rock*'s applicability in $M^{\mathcal{A}}_{drift}$ has reduced from that in $M^{\mathcal{A}}_{init}$ as $\mathcal{A}$ can no longer sample rocks in situations where the battery is half charged but needs a fully charged battery to be able to execute the action. In such scenarios, instead of relying on observation traces, our method identifies traces containing indications of actions that were affected either in their precondition or effect, discovers additional salient *pal-tuples* that were potentially affected, and adds them to the set of potentially affected *pal-tuples* Γ_δ.

To find *pal-tuples* corresponding to reduced functionality of the agent, we place the agent in an optimal planning mode and assume limited availability of observation traces $\mathbb{O}$ in the form of optimal unit-cost state-action trajectories $\langle s_0, a_1, s_1, a_2, \dots, s_{n-1}, a_n, s_n \rangle$. We generate optimal plans using $M^{\mathcal{A}}_{init}$ for all pairs of states in $\mathbb{O}$. We hypothesize that, if for a pair of states, the plan generated using $M^{\mathcal{A}}_{init}$ is shorter than the plan observed in $\mathbb{O}$, then some functionality of the agent has reduced.

Our method performs comparative analysis of optimality of the observation traces against the optimal solutions generated using $M^{\mathcal{A}}_{init}$ for same pairs of initial and final states. To begin with, we extract all the continuous state sub-sequences from $\mathbb{O}$ of the form $\langle s_0, s_1, \dots, s_n \rangle$ denoted by $\mathbb{O}_{drift}$ as they are all optimal. We then generate a set of planning problems $\mathcal{P}$ using the initial and final states of trajectories in $\mathbb{O}_{drift}$. Then, we provide the problems in $\mathcal{P}$ to $M^{\mathcal{A}}_{init}$ to get a set of optimal trajectories $\mathbb{O}_{init}$. We select all the pairs of optimal trajectories of the form $\langle o_{init}, o_{drift} \rangle$ for further analysis such that the length of $o_{init} \in \mathbb{O}_{init}$ for a problem is shorter than the length of $o_{drift} \in \mathbb{O}_{drift}$ for the same problem. For all such pairs of optimal trajectories, a subset of actions in each $o_{init} \in \mathbb{O}_{init}$ were likely affected due to the model drift. We focus on identifying the first action in each $o_{init} \in \mathbb{O}_{init}$ that was definitely affected.

To identify the affected actions, we traverse each pair of optimal trajectories $\langle o_{init}, o_{drift} \rangle$ simultaneously starting from the initial states. We add all the *pal-tuples* corresponding to the first differing action in o_{init} to Γ_δ. We do this because there are only two possible explanations for why the action differs: (i) either the action in o_{init} was applicable in a state using $M^{\mathcal{A}}_{init}$ but has become inapplicable in the same state in $M^{\mathcal{A}}_{drift}$, or (ii) it can no longer achieve the same effects in $M^{\mathcal{A}}_{drift}$ as $M^{\mathcal{A}}_{init}$. We also discover the first actions that are applicable in the same states in both the trajectories but result in different states. The effect of such actions has certainly changed in $M^{\mathcal{A}}_{drift}$. We add all the *pal-tuples* corresponding to such actions to Γ_δ. In the next section, we describe our approach to infer the correct modes of *pal-tuples* in Γ_δ.

4.2 Investigating Affected pal-tuples

This section explains how the correct modes of *pal-tuples* in Γ_δ are inferred (line 2 onwards of Alg.1). Alg. 1 creates an abstract model in which all the *pal-tuples* that are predicted to have been affected are set to ? (line 2). It then iterates over all *pal-tuples* with mode ? (line 4).

Removing inconsistent models Our method generates candidate abstract models and then removes the abstract models that are not consistent with the agent (lines 7-18 of Alg. 1). For each *pal-tuple* $\gamma \in \Gamma$, the algorithm computes a set of possible abstract models $\mathcal{M}_{abs}$ by assigning the three mode variants $+$, $-$, and $\emptyset$ to the current *pal-tuple* γ in model M_{abs} (line 6). Only one model in $\mathcal{M}_{abs}$ corresponds to the agent's updated functionality.

If the action γ_a in the *pal-tuple* γ is present in the set of action triplets generated using $\mathbb{O}$, then the pre-state of that action s_{pre} is used to create a state s_I (lines 9-10). s_I is created by removing the literals corresponding to predicate γ_p from s_{pre}. We then create a query $Q = \langle s_I, \langle \gamma_a \rangle \rangle$ (line 10), and pose it to the agent $\mathcal{A}$ (line 11). The three models are then sieved based on the comparison of their responses to the query Q with that of $\mathcal{A}$'s response θ to Q (line 12). We use the same mechanism as AIA for sieving the abstract models.

If the action corresponding to the current *pal-tuple* γ being considered is not present in any of the observed action triplets, then for every pair of abstract models in $\mathcal{M}_{abs}$ (line 14), we generate a query Q using a planning problem (line 15). We then pose the query Q to the agent (line 16) and receive its response θ. We then sieve the abstract models by asking them the same query and discarding the models whose responses are not consistent with that of the agent (line 17). The planning problem that is used to generate the query and the method that checks for consistency of abstract models' responses with that of the agent are used from AIA.

Finally, all the models that are not consistent with the agent's updated functionality are removed from the possible set of models $\mathcal{M}_{abs}$. The remaining models are returned by the algorithm. Empirically, we find that only one model is always returned by the algorithm.

4.3 Correctness

We now show that the learned drifted model representing the agent's updated functionality is consistent as defined in Def. 8 and Def. 9. The proof of the theorem is available in the extended version of the paper (Nayyar, Verma, and Srivastava 2022).

Theorem 1. Given a set of observation traces $\mathbb{O}$ generated by the drifted agent $\mathcal{A}_{drift}$, a set of queries Q posed to $\mathcal{A}_{drift}$ by Alg. 1, and the model $M^{\mathcal{A}}_{init}$ representing the agent's functionality prior to the drift, each of the models $M = \langle P, A \rangle$ in $\mathcal{M}^{\mathcal{A}}_{drift}$ learned by Alg. 1 are *consistent* with respect to all the observation traces $o \in \mathbb{O}$ and query-responses $\langle q, \theta \rangle$ for all the queries $q \in Q$.

There exists a finite set of observations that if collected will allow Alg. 1 to achieve 100% correctness with any amount of drift: this set corresponds to observations that allow line 1 of Alg. 1 to detect a change in the functionality. This includes an action triplet in an observation trace hinting at increased functionality, or a shorter plan using the previously known model hinting at reduced functionality. Thus, models learned by DAAISy are guaranteed to be completely correct irrespective of the amount of the drift if such a finite set of observations is available. While using queries significantly reduces the number of observations required, asymptotic guarantees subsume those of passive model learners while ensuring convergence to the true model.

5 Empirical Evaluation

In this section, we evaluate our approach for assessing a black-box agent to learn its model using information about its previous model and available observations. We implemented the algorithm for DAAISy in Python[1] and tested it on six planning benchmark domains from the International Planning Competition (IPC)[2]. We used the IPC domains as the unknown drifted models and generated six initial domains at random for each domain in our experiments.

To assess the performance of our approach with increasing drift, we employed two methods for generating the initial domains: (a) dropping the *pal-tuples* already present, and (b) adding new *pal-tuples*. For each experiment, we used both types of domain generation. We generated different initial models by randomly changing modes of random *pal-tuples* in the IPC domains. Thus, in all our experiments an IPC domain plays the role of ground truth M^*_{drift} and a randomized model is used as $M^{\mathcal{A}}_{init}$.

We use a very small set of observation traces $\mathbb{O}$ (single observation trace containing 10 action triplets) in all the experiments for each domain. To generate this set, we gave the agent a random problem instance from the IPC corresponding to the domain used by the agent. The agent then used Fast Downward (Helmert 2006) with LM-Cut heuristic (Helmert and Domshlak 2009) to produce an optimal solution for the given problem. The generated observation trace is provided to DAAISy as input in addition to a random $M^{\mathcal{A}}_{init}$ as discussed in Alg. 1. The exact same observation trace is used in all experiments of the same domain, without the knowledge of the drifted model of the agent, and irrespective of the amount of drift.

We measure the final accuracy of the learned model $M^{\mathcal{A}}_{drift}$ against the ground truth model M^*_{drift} using the measure of model difference $\Delta(M^{\mathcal{A}}_{drift}, M^*_{drift})$. We also measure the number of queries required to learn a model with significantly high accuracy. We compare the efficiency of DAAISy (our approach) with the Agent Interrogation Algorithm (AIA) (Verma, Marpally, and Srivastava 2021) as it is the most closely related querying-based system.

All of our experiments were executed on 5.0 GHz Intel i9 CPUs with 64 GB RAM running Ubuntu 18.04. We now discuss our results in detail below.

5.1 Results

We evaluated the performance of DAAISy along 2 directions; the number of queries it takes to learn the updated model $M^{\mathcal{A}}_{drift}$ with increasing amount of drift, and the correctness of the model $M^{\mathcal{A}}_{drift}$ it learns compared to M^*_{drift}.

Efficiency in number of queries As seen in Fig. 2, the computational cost of assessing each agent, measured in terms of the number of queries used by DAAISy, increases as the amount of drift in the model M^*_{drift} increases. This is expected as the amount of drift is directly proportional to the number of *pal-tuples* affected in the domain. This increases the number of *pal-tuples* that DAAISy identifies as affected as well as the number of queries as a result. As demonstrated in the plots, the standard deviation for number of queries remains low even when we increase the amount of drift, showing the stability of DAAISy.

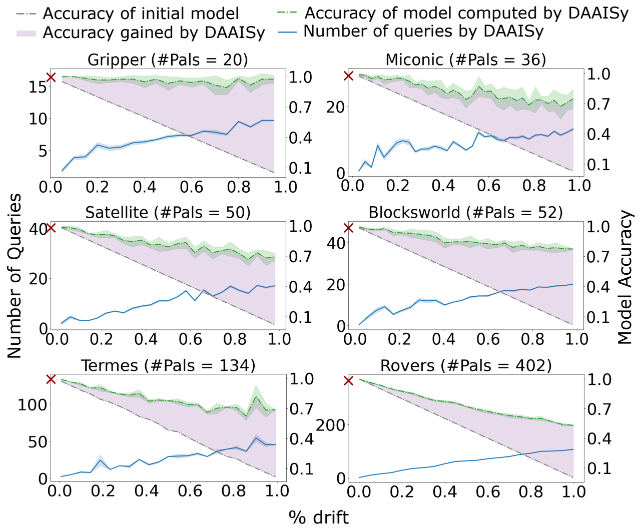

Figure 2: The number of queries used by DAAISy (our approach) and AIA (marked × on y-axis), as well as accuracy of model computed by DAAISy with increasing amount of drift. Amount of drift equals the ratio of drifted *pal-tuples* and the total number of *pal-tuples* in the domains (nPals). The number of action triplets in the observation trace used for each domain is 10.

Comparison with AIA Tab. 3 shows the average number of queries that AIA took to achieve the same level of accuracy as our approach for 50% drifted models, and DAAISy requires significantly fewer queries to reach the same levels of accuracy compared to AIA. Fig. 2 also demonstrates that DAAISy always takes fewer queries as compared to AIA to reach reasonably high levels of accuracy.

This is because AIA does not use information about the previously known model of the agent and thus ends up querying for all possible *pal-tuples*. DAAISy, on the other hand, predicts the set of *pal-tuples* that might have changed based on the observations collected from the agent and thus requires significantly fewer queries.

Correctness of learned model DAAISy computes models with at least 50% accuracy in all six domains even when they have completely drifted from their initial model, i.e., $\Delta(M^{\mathcal{A}}_{drift}, M^*_{drift}) = nPals$. It attains nearly accurate models for Gripper and Blocksworld for upto 40% drift. Even in scenarios where the agent's model drift is more than 50%, DAAISy achieves at least 70% accuracy in five domains. Note that DAAISy is guaranteed to find the correct mode for an identified affected *pal-tuple*. The reason for less than 100% accuracy when using DAAISy is that it does not predict a *pal-tuple* to be affected unless it encounters an obser-

[1] Code available at https://github.com/AAIR-lab/DAAISy

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

Domain	#Pals	AIA	DAAISy
Gripper	20	15.0	6.5
Miconic	36	32.0	7.7
Satellite	50	34.0	9.0
Blocksworld	52	40.0	11.4
Termes	134	115.0	27.0
Rovers	402	316.0	61.0

Table 3: The average number of queries taken by AIA to achieve the same level of accuracy as DAAISy (our approach) for 50% drifted models.

vation trace conflicting with $M^{\mathcal{A}}_{init}$. Thus, the learned model $M^{\mathcal{A}}_{drift}$, even though consistent with all the observation traces, may end up being inaccurate when compared to M^{*}_{drift}.

Discussion AIA always ends up learning completely accurate models, but as noted above, this is because AIA queries exhaustively for all the *pal-tuples* in the model. There is a clear trade-off between the number of queries that DAAISy takes to learn the model as compared to AIA and the correctness of the learned model. As evident from the results, if the model has not drifted much, DAAISy can serve as a better approach to efficiently learn the updated functionality of the agent with less overhead as compared to AIA. Deciding the amount of drift after which it would make sense to switch to querying the model from scratch is a useful analysis not addressed in this paper.

6 Related Work

White-box model drift Bryce, Benton, and Boldt (2016) address the problem of learning the updated mental model of a user using particle filtering given prior knowledge about the user's mental model. However, they assume that the entity being modeled can tell the learning system about flaws in the learned model if needed. Eiter et al. (2005, 2010) propose a framework for updating action laws depicted in the form of graphs representing the state space. They assume that changes can only happen in effects, and that knowledge about the state space and what effects might change is available beforehand. Our work does not make such assumptions to learn the correct model of the agent's functionalities.

Action model learning The problem of learning agent models from observations of its behavior is an active area of research (Gil 1994; Yang, Wu, and Jiang 2007; Cresswell, McCluskey, and West 2009; Zhuo and Kambhampati 2013; Arora et al. 2018; Aineto, Celorrio, and Onaindia 2019). Recent work addresses active querying to learn the action model of an agent (Rodrigues et al. 2011; Verma, Marpally, and Srivastava 2021). However, these methods do not address the problem of reducing the computational cost of differential model assessment, which is crucial in non-stationary settings.

Online action model learning approaches learn the model of an agent while incorporating new observations of the agent behavior (Čertický 2014; Lamanna et al. 2021a,b). Unlike our approach, they do not handle cases where (i) the new observations are not consistent with the older ones due to changes in the agent's behavior; and/or (ii) there is reduction in functionality of the agent. Lindsay (2021) solve the problem of learning all static predicates in a domain. They start with a correct partial model that captures the dynamic part of the model accurately and generate negative examples by assuming access to all possible positive examples. Our method is different in that it does not make such assumptions and leverages a small set of available observations to infer about increased and reduced functionality of an agent's model.

Model reconciliation Model reconciliation literature (Chakraborti et al. 2017; Sreedharan et al. 2019; Sreedharan, Chakraborti, and Kambhampati 2021) deals with inferring the differences between the user and the agent models and removing them using explanations. These methods consider white-box known models whereas our approach works with black-box models of the agent.

7 Conclusions and Future Work

We presented a novel method for *differential assessment* of black-box AI systems to learn models of true functionality of agents that have drifted from their previously known functionality. Our approach provides guarantees of correctness w.r.t. observations. Our evaluation demonstrates that our system, DAAISy, efficiently learns a highly accurate model of agent's functionality issuing a significantly lower number of queries as opposed to relearning from scratch. In the future, we plan to extend the framework to more general classes, stochastic settings, and models. Analyzing and predicting switching points from selective querying in DAAISy to relearning from scratch without compromising the correctness of the learned models is also a promising direction for future work.

Acknowledgements

We thank anonymous reviewers for their helpful feedback on the paper. This work was supported in part by the NSF under grants IIS 1942856, IIS 1909370, and the ONR grant N00014-21-1-2045.

References

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

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

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

Bryce, D.; Benton, J.; and Boldt, M. W. 2016. Maintaining Evolving Domain Models. In *Proc. IJCAI*.

Čertický, M. 2014. Real-Time Action Model Learning with Online Algorithm 3SG. *Applied Artificial Intelligence*, 28(7): 690–711.

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

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

Eiter, T.; Erdem, E.; Fink, M.; and Senko, J. 2005. Updating Action Domain Descriptions. In *Proc. IJCAI*.

Eiter, T.; Erdem, E.; Fink, M.; and Senko, J. 2010. Updating Action Domain Descriptions. *Artificial Intelligence*, 174(15): 1172–1221.

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

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

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

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

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

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

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

Lamanna, L.; Gerevini, A. E.; Saetti, A.; Serafini, L.; and Traverso, P. 2021a. On-line Learning of Planning Domains from Sensor Data in PAL: Scaling up to Large State Spaces. In *Proc. AAAI*.

Lamanna, L.; Saetti, A.; Serafini, L.; Gerevini, A.; and Traverso, P. 2021b. Online Learning of Action Models for PDDL Planning. In *Proc. IJCAI*.

Lindsay, A. 2021. Reuniting the LOCM Family: An Alternative Method for Identifying Static Relationships. In *ICAPS 2021 KEPS Workshop*.

Long, D.; and Fox, M. 2003. The 3rd International Planning Competition: Results and Analysis. *Journal of Artificial Intelligence Research*, 20: 1–59.

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

Nayyar, R. K.; Verma, P.; and Srivastava, S. 2022. Differential Assessment of Black-Box AI Agents. *arXiv preprint arXiv: 2203.13236*.

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

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

Sreedharan, S.; Chakraborti, T.; and Kambhampati, S. 2021. Foundations of Explanations as Model Reconciliation. *Artificial Intelligence*, 103558.

Sreedharan, S.; Hernandez, A. O.; Mishra, A. P.; and Kambhampati, S. 2019. Model-Free Model Reconciliation. In *Proc. IJCAI*.

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

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

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

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

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