BEN-GURION UNIVERSITY OF THE NEGEV
FACULTY OF ENGINEERING SCIENCES

DEPARTMENT OF SOFTWARE AND INFORMATION SYSTEMS ENGINEERING

Online Learning of Action Models with Optimized Decision Making

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE M.SC DEGREE

By: Omer Eliyahu , Roni Stern, Argaman Mordoch
{mordocha, omereliy}@post.bgu.ac.il, roni.stern@gmail.com

December 2024

BEN-GURION UNIVERSITY OF THE NEGEV

FACULTY OF ENGINEERING SCIENCES

DEPARTMENT OF SOFTWARE AND INFORMATION SYSTEMS ENGINEERING

Online Learning of Action Models with Optimized Decision Making

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE M.SC DEGREE

BY: OMER ELIYAHU

SUPERVISED BY: PROF. RONI STERN

AUTHOR:	....	Date: ....
SUPERVISOR:	....	Date: ....
SUPERVISOR:	....	Date: ....
CHAIRMAN OF GRADUATE STUDIES COMMITTEE:	....	Date: ....

December 2024

Abstract

Automated planning enables intelligent agents to reason about actions, predict outcomes, and generate sequences of steps to achieve specified goals. Central to this process are **action models**, which define the preconditions and effects of actions within a domain. Traditionally, these models are manually crafted by experts or derived from offline learning on pre-collected plan traces. However, manual specification is time-consuming, prone to errors and demands a wide knowledge of the domain. While pre-collected plan traces are not always available, and must often be collected by trial-and-error, purely offline approaches can fail.

This thesis introduces a novel method for decision-making in *online* learning, prioritizing actions that reduce uncertainty and refine the **hypothesis space** efficiently. By incorporating both positive and negative demonstrations, the proposed algorithm ensures robust learning while accelerating convergence to correct and comprehensive models. Building on existing frameworks such as OLAM (Online Learning of Action Models), this work demonstrates active action taking approach for learning action models, hopefully paving the way for more adaptable and autonomous planning systems.

Introduction

Automated planning enables intelligent agents to construct sequences of actions to achieve desired goals. Planning systems conventionally depend on domain models, which define actions that consist of **preconditions**, Facts that must be true for the action to be valid, and **effects**, who can be positive or negative and specify the Facts that become true and false, respectively, once the action is performed. Although effective in static environments, the manual specification of such models is often inefficient and error-prone, requiring substantial domain expertise. As an alternative, learning-based methods (Amir and Chang, 2008; Zhuo and Kambhampati, 2013) have emerged to address these limitations. A promising direction is to have autonomous agents *learn* action models. By interacting with the environment (or from collected plan traces), the agent refines hypotheses about preconditions and effects. This action model learning is especially critical in domains where:

1. The world dynamics are initially unknown or only partially specified.

2. Collecting expert-specified domain models is infeasible or too costly.

Early work in the field has shown that learning action models can significantly reduce the burden of domain engineering and improve an agent's ability to adapt to new or changing environments. For example, SAM (by Stern and colleagues (Juba et al., 2021)) applies a logical approach to learning action models, focusing on incremental refinement of partial models. online methods, such as Online Learning of Action Models (OLAM) (lamanna et al., 2021), demonstrate how agents can *actively* generate informative plan traces. Furthermore, theoretical guarantees have been explored in the line of research on **Learning action models with guarantees** (Aineto and Scala, 2024), ensuring that the learned model converges to correctness under certain assumptions.

In this paper, we propose a new online learning method that leverages the strengths of these prior approaches—particularly OLAM's iterative *plan-observe-update* process—and introduces a novel strategy to maximize knowledge gain.

Chapter 1

Background and Related Work

In this section, we review the relevant background and related work. Specifically, we provide background on automated planning and learning action models.

1.1 Automated Planning

In this work, we focus on *classical planning* problems. A classical planning domain, denoted D, is defined by a tuple $\langle T, \mathcal{F}, \mathcal{A}, M \rangle$ where T is a set of types, $\mathcal{F}$ is a set of lifted fluents, $\mathcal{A}$ is a set of lifted actions, and M is an action model for $\mathcal{A}$. A classical planning problem, denoted as Π, is defined by a tuple $\langle D, O, s_I, G \rangle$ where D is a classical planning domain, O is a set of objects, s_I is the initial state (i.e., the state of the world before planning), G is a set of grounded literals that define when the goal has been found. solution to a planning problem is a sequence of grounded actions that can be applied to s_I and, if applied to s_I results in a state s_G that contains all grounded literals in G. This sequence of grounded actions is called a plan and is denoted as π. The trajectory of a plan starts with s_I and ends with a goal state s_G (where $G \subseteq sG$).(Juba et al., 2021)

1.1.1 Planning Domain Definition Language

The Planning Domain Definition Language (PDDL) is a formal language for describing classical planning problems. Figure 1.1 shows an example of a PDDL problem.

```
(define (problem truck-problem)
    (:domain truck-domain)

    (:objects
        truck1 - truck
        package1 - package
        A B C - location
    )

    (:init
        (at truck1 A)
        (at package1 A)
    )

    (:goal
        (at package1 C)
    )
)
```

FIGURE 1.1: PDDL planing problem of Initial state where a truck and a package are in some location, 'A', and the goal is to achieve a state where the package is in location 'C'.

PDDL problem solvers

There are many algorithms for solving PDDL problems, for example *FastDownward*(Helmert, 2006,) a highly efficient planner that uses heuristic search, particularly landmark-cut heuristic, to optimize planning in large state spaces. *FastForward (FF)*(Hoffmann, 2001), Uses relaxed plan heuristics to generate plans quickly and is particularly effective in deterministic environments. *LAMA* (Richter and Westphal, 2010), An anytime planner that utilizes cost-sensitive heuristics to iteratively improve solution quality. *SATPlan*(Kautz and Selman, 1992), A planner that encodes the planning problem into a Boolean satisfiability (SAT) problem, solving it using SAT solvers.

1.2 Action Model Learning

Automated planning is based on a well-defined *action model*, which specifies the preconditions and effects of each action within a domain. However, manually defining these models can be costly, error prone, and infeasible for complex environments. To address this challenge, *Action Model Learning* (AML) aims to automatically infer action models from observed transitions. enabling an adaptive and efficient approach to address those problems.

There are several algorithms for learning action models, each differing in their assumptions about available data, observability conditions, and learning strategies. This section reviews key approaches in action model learning, highlighting their advantages and limitations.

Definition 1.2.1 (transition)**.** Let a be an action, s be the a state , A triplet (s, a, s') is called a correct transition if s' is a result of correctly applying a from state s.

Action model learning algorithms are designed to acquire knowledge of the action model in the domain using observations of transitions that can occur under various conditions: *Full observability*, The agent sees all relevant parts of the state of the environment. *Partial observability*, Some aspects of the state remain hidden or unobserved. *Noisy data*, Observations and action outcomes may be affected by uncertainty or sensor errors. *Action-only observations*, Only the executed actions are known, with limited or no direct visibility of how the state changes.

Algorithms for action model learning can be partitioned into offline algorithms and online algorithms. We describe each in the following.

1.3 Offline Learning Approaches

Offline learning methods require a *pre-collected* set of plan traces, each trace being a sequence of states and actions executed, before the learning process begins (Juba et al., 2021; Aineto et al., 2019). For example, Aineto et al. (2018) and Aineto et al. (2019) propose **FAMA**, which encodes the learning task itself as a classical planning problem. Such offline learners can handle partial plan information or incomplete models, but face two main drawbacks:

1. **Dependence on Traces.** Gathering sufficiently rich plan traces in unknown or dynamic environments can be difficult (Fern et al., 2004).

2. **Limited Exploration.** If no trace covers a specific state transition, the learner might not discover an essential precondition or effect and has no option of continuous exploration of the action model (Cresswell et al., 2013).

The SAM learning algorithm is another offline learning algorithm. SAM works by eliminating possible preconditions when transitions occur without them. and accumulates delete/add-effects when observing changes between the pre-state and post-state. Provides a special form of guarantee that is referred to as a "safe action model". A limitation of the SAM learning algorithm is that it only guarantees safety in domains where *injective binding assumption* is true. This assumption means that a specific object may only be bound to a single argument of an action. An example where *injective binding assumption* does not hold: let $say - hellow(m1 - person, m2 - person)$ be an action that takes as an argument 2 persons. So, in the following grounding: $say - hello(man_1, man_1)$, man_1 is bound to two arguments. The Extended-SAM learning algorithm (ESAM) generalizes the SAM learning algorithm to address domains in which the *injective binding assumption* does not hold.

Additional line of research explores the learning of action models with formal guarantees, as highlighted by Aineto and Scala (2024). In this paradigm, the learning task is formulated within a *version space*, a set of hypothesized domain models that remain consistent with all observed transitions. By methodically pruning any hypothesis contradicted by new data, the learner refines and maintains a compact representation of every solution that still fits the evidence. This approach reveals a *spectrum* of candidate action models — which range from more 'pessimistic' (sound) under-approximations to more 'optimistic' (complete) over-approximations, each of which supports different reasoning capabilities while convergence to the true model is ongoing.

MACQ is a recently proposed framework for running and evaluating action model learning algorithms. In this framework, we can generate training trajectories and run some of the learning algorithms. However, MACQ has several limitations. First, it does not include the SAM and ESAM algorithms. Second, it does not support direct comparison of the algorithms in it, since it does not include any action model learning metrics. This is because such comparison is not trivial as the action model learning algorithm tends to return various types of models. These models can differ syntactically (for example, the name of the action). The problem will be addressed later in this paper as we propose a metric for the evaluation of different action models learned from the same domain.

1.4 Online Learning Approaches

In an online setting, the agent learns incrementally by *executing actions* in the environment and by observing their results (Gil, 1994b; Gil, 1994a; Wang, 1996). This approach integrates planning and learning in a feedback loop.

1. *Plan* with the current model to choose actions.
2. *Execute* those actions in the real or simulated environment.
3. *Observe* the resulting state changes.
4. *Update* the model to be more accurate or complete.

Examples include instance-based relational model learning (Xu and Laird, 2010) and probabilistic action models (Certicky, 2014), which reduce the reliance on comprehensive offline data. However, a key challenge is deciding which actions (and goals) to explore to gain the most information (Rodrigues et al., 2010). Random exploration can be slow or may overlook crucial transitions.

A prominent online approach is the Online Learning of Action Models (OLAM) algorithm, introduced by lamanna et al. (2021). OLAM specifically tackles the generation of *informative plan traces online*, interleaving:

- **Planning and Execution**: The agent selects goal states that by planning to that state, assures obtaining some missing knowledge of the actions executed at the plan.
- **Model Refinement**: After each plan is executed, the algorithm updates the set of known conditions for each action (removing contradictions and adding newly discovered requirements).

Crucially, OLAM provides *theoretical guarantees* of convergence to a correct and integral model (with full observability).

1.5 Relation to Our Work

Our framework draws heavily on:

1. OLAM's iterative *plan-observe-update* structure (lamanna et al., 2021),
2. SAM framework symbolic reasoning approach to action model learning (Juba et al., 2021),

3. Version-space guarantees from Aineto and Scala (2024).

We combine these ideas to propose an online algorithm that specifically maximizes knowledge gain at each decision point. Rather than focusing solely on safe or complete hypotheses, we choose actions that reduce domain uncertainty most efficiently, an approach that we believe can yield faster convergence in complex domains. The next chapters describe our approach in detail, including:

- The **theoretical foundations** for action model learning.
- A **knowledge gain metric** based on the portion of hypotheses that is ruled out in any iteration out of the hypothesis space.
- Empirical demonstrations on different planning benchmarks.

Together, these contributions aim to advance the capabilities of online action model learning, building on and extending the successes of OLAM, SAM, and guaranteed learning methods.

Definition[section]

1.6 Definitions

Notation and Basic Concepts:

Let O be an ordered set of objects, and a be a lifted action.

Definition 1.6.1 (grounding). $\langle a, O \rangle$ is a *grounding* of a if O satisfies $\mathrm{param}(a)$ (the parameters of a).

Let L_a be the parameter-bound literals of an action a. s, s', Pre-state and post-state, respectively.

Definition 1.6.2 (binding). Given f is a set of *grounded* literals, returns $bindP(f, O)$, parameter-bound literals in respect to O. $bindP^{-1}(F, O)$, Given F is a set of parameter-bound literals, returns groundings of F respecting the order of O.

$pre^*(a), eff^{*-}(a), eff^{*+}(a)$: The *REAL* Action model preconditions / effects of a.

Initialization: For an action a, we define:
$pre(a)$ - all predicates initially not ruled out as a precondition,
$pre_?(a)$ - CNF formula representing uncertain preconditions,
$eff^+(a)$ - observed predicates that are *surely* add effects,

$eff^{-}(a)$ - observed predicates that are *surely* remove effects,

$eff^{?+}(a)$ - predicates that are possibly an add effect,

$eff^{?-}$ -predicates that are possibly a remove effect

Chapter 2

Methodology

In this section, we describe the research question and objectives of the proposed research, and the planned methodology for answering these questions and achieving the research objectives.

2.0.1 Research questions and objectives

Objective:

1. Develop an action model learning comparison paradigm and implement it in MACQ.
2. Develop an online version of ESAM and compare it with existing online action model learning algorithms.

Research questions: 1. How to properly evaluate and compare different action model learning algorithms? 2. How can ESAM be used to learn an action model in an online action model learning environment?

2.1 A Paradigm for Comparing Action Model Learning Algorithms

Every action model learning algorithm can be viewed as a function that accepts a set of trajectories from the original domain and a learned domain. so by comparing the coherence of an action applicability [[roni: what is "coherence of an action applicability"?]] in the original domain and its matching in the learned domain. The result is a scaling of *safety* and *completeness* in the learned domain. [[roni: "is a scaling of" does not make sense]] Comparing action model learning algorithms is challenging because different algorithms may output models with varied syntactic structures, representations, or levels of abstraction As mentioned above, learned domains can differ syntactically from the original domain, making direct comparisons difficult. To mitigate this, we introduce an **encoder-decoder** mechanism: The **decoder** maps actions in the learned model m back to the original domain language m^*. The encoder **encoder** transforms the original domain m^* into the representation format of the learned model m. This process ensures that comparisons are done in a normalized format, reducing discrepancies caused by syntactic variations. Let us formally describe the encode and decode functions:

- $encode : A_{m^*} \rightarrow P(A_m)$
  and $encode(s, a, m^*, m) = \{a' \in m | a' \; is \; a \; proxy \; action \; of \; a. \}$
- $decode : A_m \rightarrow A_{m^*}$ and $decode(s, a', m^*, m) = a^* \in A_{m^*}$ s.t. a^* is a proxy of a.

using these functions, we can now describe and use the following evaluation metrics.

Evaluation Metrics

To systematically compare the learned action model m with the ground truth model m^*, we define the following metrics:

True Positives (TP): The set of instances where for every pre-state s and action $a \in A_{m^*}$, if a is applicable in s, then at least one proxy action $a \in encode(s, a, m^*, m)$ is applicable in s.

False Positives (FP): The set of instances where there exists a proxy action $a \in A_m$ that is applicable in a pre-state s, but there is no corresponding action $a^* \in A_{m^*} \;\; s.t. \; a^* = decode(s, a', m^*, m)$ and a^* is applicable in s.

False Negatives (FN): The set of instances where for some pre-state s and action $a^* \in A_{m^*}$ $s.t.$ a^* is applicable in s, but no proxy action $a \in encode(s, a, m^*, m)$ is applicable in s.

True Negatives (TN): The set of instances where for every pre-state s, if a proxy action $a \in A_m$ is not applicable in s, then there is no corresponding action $a* \in A_{m^*}$ such that a is applicable in s.

Action Applicability	**Original Model** m^*	**Learned Model** m
True Positive (TP)	Applicable	Applicable
False Positive (FP)	Not Applicable	Applicable
True Negative (TN)	Not Applicable	Not Applicable
False Negative (FN)	Applicable	Not Applicable

TABLE 2.1: Comparison of action applicability in the original and learned models.

These metrics allow us to quantify the *precision* and *recall*, where *precision* is $\frac{TP}{TP+FP}$ and *recall* is $\frac{TP}{TP+FN}$ This quantification will be useful when choosing the action model learning method based on restrictions and use cases of the learned model.These metrics provide a comprehensive evaluation of the learned model's ability to replicate the original model's behavior.

The *semantic* precision and recall of the preconditions and effects are computed with respect to a set of trajectories that we call the *test trajectories*. A reasonable choice for how to create these test trajectories is by running a planner on a set of test problems with the real action model. Alternatively, one may run a random walk with real action model, starting from random start states. To compute the defined above metrics, we compute the TP, TN, FP, and FN of every transition in the test trajectories.

In the proposed research, we will implement this paradigm and an encoder-decoder for algorithms X, Y, Z. Then, we will evaluate them and compare....

2.2 Online Action Model Learning with SAM

In this part of the proposed research, we will develop an online approach to learn action models without any given trajectories but by letting an agent perform action solely to learn the action model while using as little interaction as possible. Next, we describe the details of the different elements of this solution.

2.2.1 Update Rules

2.2.2 Action a *is* Applicable in State s

Suppose a is successfully applied in s, resulting in s'. Denote this by $\mathrm{app}(a, s) = s'$. Then we update:

- $pre_?(a) \leftarrow pre_?(a) \wedge \bigwedge_{l \in pre(a) \setminus bindP(s,O)} (\neg x_l)$

- $pre(a) \;\leftarrow\; pre(a) \;\cap\; bindP(s, O)$
- $eff^{+}(a) \;\leftarrow\; eff^{+}(a) \cup bindP(s' \setminus s,\, O)$ (*surely add* effect)
- $eff^{-}(a) \;\leftarrow\; eff^{-}(a) \cup bindP(s \setminus s',\, O)$ (*surely delete* effect)
- $eff^{?+}(a) \;\leftarrow\; eff^{(?+)}(a) \cap bindP(s \cap s',\, O)$
- $eff^{?-}(a) \;\leftarrow\; eff^{?-}(a) \;\setminus\; bindP\big(s \cup s',\, O\big)$

2.2.3 Action a ***is not*** Applicable in State s

If (a, O) fails to execute in s because $pre(a) \not\subseteq s$, then:

$$pre_{?}(a) \;\leftarrow\; pre_{?}(a) \;\wedge \bigvee_{l \in pre(a) \setminus bindP(s,O)} x_l.$$

In words, at least one of those missing literals must be a *true* precondition.

2.2.4 Action Choosing Methodology

2.2.5 Grounded-Action Probability of applicability

Define the indicator

$$\mathrm{app}(a, O, s) \;=\; \begin{cases} 1, & \text{if } a \text{ is applicable with arguments } O \text{ in } s, \\[1em] 0, & \text{otherwise.} \end{cases}$$

We assume that each assignment that satisfies $\mathrm{cnf}(pre_?(a))$ is equally likely. Then the **probability** of (a, O) being applicable in s is

$$\Pr[\mathrm{app}(a, O, s) = 1] \;=\; \frac{\bigl|\mathrm{SAT}\bigl(\mathrm{cnf}(pre_?(a), O, s)\bigr)\bigr|}{\bigl|\mathrm{SAT}\bigl(\mathrm{cnf}(pre_?(a))\bigr)\bigr|}.$$

2.2.6 Preconditions Gained Knowledge

When a Succeeds in s

If (a, O) is successfully applied from s, any literal in $pre(a)$ that does *not* appear in $bindP(s, O)$ is ruled out. We define:

$$\mathrm{preAppPotential}(a, O, s) \;=\; \bigl|\, pre(a) \setminus bindP(s, O) \,\bigr|.$$

When a Fails in s

If (a, O) fails, we must add a disjunction that *at least one* missing literal is truly a precondition. The knowledge gained can be measured by the fraction of the solution-space pruned from $\mathrm{cnf}(pre_?(a))$. Let

$$\mathrm{preFailPotential}(a, O, s) \;=\; 1 \;-\; \frac{\Big|\mathrm{SAT}\big(\mathrm{cnf}\big(pre_?(a)\big)\big| \;-\; \Big|\mathrm{SAT}\big(\mathrm{cnf}'\big(pre_?(a)\big)\big|}{2^{\,|L_a|}}$$

where $\mathrm{cnf}'(pre_?(a))$ is the updated CNF after adding that disjunction.

2.2.7 Effects Gained Knowledge

We track how many literals remain uncertain in $eff^{(?+)}(a)$ and $eff^{(?-)}(a)$. Once those sets are empty, we know the exact add/delete sets. We can define:

$$\mathrm{grade}\big(eff(a)\big) = \frac{|L_a| - \big|eff^{(?+)}(a) \,\cup\, eff^{(?-)}(a)\big|}{|L_a|}.$$

When applying (a, O) successfully:

$$\mathrm{eff}^{+}\mathrm{Potential}(a, O, s) = \frac{\big|eff^{(?+)}(a) \,\setminus\, bindP(s, O)\big|}{|L_a|}$$

$$\text{eff}^{-}\text{Potential}(a, O, s) = \frac{\left|eff^{(?-)}(a) \cap bindP(s, O)\right|}{|L_a|}$$

Therefore the total knowledge gained about the effects is:

$$\text{effPotential}(a, O, s) = \ \text{eff}^{+}\text{Potential}(a, O, s) \ + \ \text{eff}^{-}\text{Potential}(a, O, s).$$

2.2.8 Reward Calculation

Define:

$$\text{sucPotential}(a, O, s) \ = \ \text{effPotential}(a, O, s) \ + \ \text{preAppPotential}(a, O, s).$$

Then, if $\text{app}(a, O, s) = 1$ (success), we expect to gain sucPotential.

if $\text{app}(a, O, s) = 0$ (success), we expect to gain preFailPotential. So:W

$$f(a, O, s) = \begin{cases} \text{sucPotential}(a, O, s), & \text{if } \text{app}(a, O, s) = 1, \\ \text{preFailPotential}(a, O, s), & \text{otherwise}. \end{cases}$$

Overall, the expected reward is:

$$\mathbb{E}\big[X_{(a,O,s)}\big] = \Pr[\mathrm{app}(a,O,s)=1]\cdot \mathrm{sucPotential}(a,O,s)$$

$$+\ \Pr[\mathrm{app}(a,O,s)=0]\cdot \mathrm{preFailPotential}(a,O,s).$$

2.2.9 Knowledge-Gain-Motivated Action Choosing

We present Two main strategies for choosing a single action to be executed immediately:

1. **Greedy Choice:**
$$\max_{(a,O)\,\in\,\mathrm{GroundedActions}(s)} \Big\{\ \mathbb{E}[X_{(a,O,s)}]\Big\}.$$
Pick the (a,O) that maximizes the expected knowledge gain.

2. **Weighted choices:**
Let $w(a,O,s)=\mathbb{E}[X_{(a,O,s)}]$. Then choose (a,O) with probability
$$P_s\big(\mathrm{choose}(a,O)\big)\ =\ \frac{w(a,O,s)}{\sum_{(a',O')} w(a',O',s)}.$$

Some states may not have the potential to acquire knowledge immediately. In this case, we will want to plan using the knowledge gained at the point s.t.

The plan has the highest probability of successfully applying the knowledge and will result in a state where some knowledge will be gained. at the point well use a naive choice for a certain depth of plan s.t. the expected reward (i.e the knowledge gained) is sufficient. Some question may arise from planning under uncertainty of the applicability of the action. future work of combining RL methods for learning action models hopefully will address those questions.

2.2.10 Algorithm

Algorithm 1 Overall Learning Algorithm

Input: $(s,\ \text{Actions},\ \text{Objects})$

Output: M, the updated action model

```
1  Initialization: $pre(a) \leftarrow L_a$,  $pre_?(a) \leftarrow \text{True}$,  $eff^+(a) \leftarrow \varnothing$,  $eff^-(a) \leftarrow \varnothing$,
   $eff^{(?+)}(a) \leftarrow L_a$,  $eff^{(?-)}(a) \leftarrow L_a$,
   $M \leftarrow \big(\text{Actions}, \{\, pre(a) | a \in Action\}, \{\, (eff^+(a),\ eff^-(a)) | a \in Action\}\big)$.
2  while $\exists\,(a,O)$ s.t. $\Pr[\text{app}(a,O,s)=1] > 0$ do
3      if $\exists\,(a,O)$ s.t. $\mathbb{E}[X_{(a,O,s)}] > 0$ then
4          groundedAction ← chooseAction$(s,\ \text{Actions})$
           execute(groundedAction)
5          if execution fails then
6              $pre_?(a) \leftarrow pre_?(a) \wedge \bigvee_{l \in \big(pre(a) \setminus bindP(s,O)\big)} x_l$
7          else
8              $pre_?(a) \leftarrow pre_?(a) \wedge \bigwedge_{l \in \big(pre(a) \setminus bindP(s,O)\big)} (\neg x_l)$
               Minimize($pre_?(a)$)
               $pre(a) \leftarrow pre(a) \cap bindP^{-1}(s,O)$
               $eff^+(a) \leftarrow eff^+(a) \cup bindP\big(s' \setminus s,\ O\big) \ldots$
9      else
10         if $\exists\, s'$ s.t. $s'$ is reachable from $s$ using M and $\exists\,(a,O) : \mathbb{E}[X_{(a,O,s')}] > 0$ then
11             plan$(M,\ s,\ s')$
12     $M \leftarrow \big(\text{Actions}, \{\, pre(a) | a \in Action\}, \{\, (eff^+(a),\ eff^-(a)) | a \in Action\}\big)$.
13 return $M$
```

Bibliography

Aineto, D., S. Jiménez, and E. Onaindia (2018). “Learning strips action models with classical planning”. In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Aineto, Diego, Sergio Jiménez Celorrio, and Eva Onaindia (2019). “Learning action models with minimal observability”. In: *Artificial Intelligence* 275, pp. 104–137. DOI: 10.1016/j.artint.2019.05.003.

Aineto, Diego and Enrico Scala (Nov. 2024). “Action Model Learning with Guarantees”. In: *Proceedings of the TwentyFirst International Conference on Principles of Knowledge Representation and Reasoning*. KR-2024. International Joint Conferences on Artificial Intelligence Organization, pp. 801–811. DOI: 10.24963/kr.2024/75. URL: http://dx.doi.org/10.24963/kr.2024/75.

Amir, Eyal and Allen Chang (2008). “Learning partially observable deterministic action models”. In: *Journal of Artificial Intelligence Research* 33, pp. 349–402. DOI: doi.org/10.48550/arXiv.1401.3437.

Certicky, Michal (2014). “Real-time action model learning with online algorithm 3SG”. In: *Applied Artificial Intelligence* 28.7, pp. 690–711. DOI: 10.1080/08839514.2014.927692.

Cresswell, Stephen, Thomas Leo McCluskey, and Margaret Mary West (2013). “Acquiring planning domain models using LOCM”. In: *The Knowledge Engineering Review* 28.2, pp. 195–213. DOI: doi:10.1017/S0269888912000422.

Fern, Alan, Sung Wook Yoon, and Robert Givan (2004). “Learning domain-specific control knowledge from random walks”. In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Gil, Yolanda (1994a). “Learning by experimentation: Incremental refinement of incomplete planning domains”. In: *Proceedings of the International Conference on Machine Learning (ICML)*. DOI: doi.org/10.1016/B978-1-55860-335-6.50019-2.

— (1994b). “Learning new planning operators by exploration and experimentation”. In: *AAAI Conference on Artificial Intelligence*. URL: https://api.semanticscholar.org/CorpusID:3088539.

Helmert, Malte (2006). “The Fast Downward planning system.” In: *Journal of Artificial Intelligence Research* 26, pp. 191–246.

Hoffmann, Jörg (Sept. 2001). "FF The Fast-Forward Planning System". In: *AI Mag.* 22.3, 57–62. ISSN: 0738-4602. DOI: 10.1609/aimag.v22i3.1572. URL: https://doi.org/10.1609/aimag.v22i3.1572.

Juba, Brendan, Hai S. Le, and Roni Stern (2021). "Safe Learning of Lifted Action Models". In: *CoRR* abs/2107.04169. arXiv: 2107.04169. URL: https://arxiv.org/abs/2107.04169.

Kautz, Henry and Bart Selman (1992). "Planning as satisfiability". In: *Proceedings of the 10th European Conference on Artificial Intelligence*. ECAI '92. Vienna, Austria: John Wiley & Sons, Inc., 359–363. ISBN: 0471936081.

Richter, S. and M. Westphal (Sept. 2010). "The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks". In: *Journal of Artificial Intelligence Research* 39, 127–177. ISSN: 1076-9757. DOI: 10.1613/jair.2972. URL: http://dx.doi.org/10.1613/jair.2972.

Rodrigues, Christophe, Pierre Gérard, and Céline Rouveirol (2010). "Incremental learning of relational action models in noisy environments". In: *Proceedings of the International Conference on Inductive Logic Programming (ILP)*. DOI: 10.1007/978-3-642-21295-6_24.

Wang, Xuemei (1996). "Planning while learning operators". In: *AAAI Conference on Artificial Intelligence*.

Xu, Joseph Z. and John E. Laird (2010). "Instance-based online learning of deterministic relational action models". In: *AAAI Conference on Artificial Intelligence*. DOI: 10.1609/aaai.v24i1.7569.

Zhuo, Hankz Hankui and Subbarao Kambhampati (2013). "Action-model acquisition from noisy plan traces". In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*.

lamanna, l., a.Saetti, l.Serafini, A.Gerevin, and p.Travers (2021). "Online Learning of Action Models for PDDL Planning". In: *Artificial Intelligence* 1, pp. 4112–4118. DOI: 10.24963/ijcai.2021/566. URL: https://doi.org/10.24963/ijcai.2021/566.