Online Learning of Action Models for PDDL Planning

Leonardo Lamanna,[1,2] **Alessandro Saetti,** [2] **Luciano Serafini,** [1]
Alfonso E. Gerevini, [2] **Paolo Traverso** [1]

[1] Fondazione Bruno Kessler (FBK), Povo, Trento, Italy
[2] Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Brescia, Italy

Abstract

The automated learning of action models is widely recognised as a key and compelling challenge to address the difficulties of the manual specification of planning domains. Most state-of-the-art methods perform this learning offline from an input set of plan traces generated by the execution of (successful) plans. However, how to generate informative plan traces for learning action models is still an open issue. Moreover, plan traces might not be available for a new environment. In this paper, we propose an algorithm for learning action models *online*, incrementally during the execution of plans. Such plans are generated to achieve goals that the algorithm decides online in order to obtain informative plan traces and reach states from which useful information can be learned. We show some fundamental theoretical properties of the algorithm, and we experimentally evaluate the online learning of the actions models over a large set of IPC domains.

Introduction

Automated planning techniques require the specification of planning domains through action models (a set of preconditions and a set of effects for each domain action). However, the manual specification of the action models is often an inaccurate, time consuming, and error-prone task. The automated learning of action models is widely recognised as a key and compelling challenge to overcome these difficulties. Several works have addressed the task of learning action models, and have provided important results from different perspectives and according to different assumptions, see, e.g., (Yang, Wu, and Jang 2007; Amir and Chang 2008; Xu and Laird 2010; Rodrigues et al. 2011; Mourão et al. 2012; Zhuo and Kambhampati 2013; Cresswell, McCluskey, and West 2013; Certicky 2014; Aineto, Jiménez, and Onaindia 2018; Aineto, Jiménez Celorrio, and Onaindia 2019).

However, most of the recent and state-of-the-art methods perform learning offline, and require as input a set of plan traces generated by previously executed actions. This has two major drawbacks. First, often agents need to learn the model of the domain *online*, because they need to explore an unknown environment, acquire information, and learn a model by experimenting the execution of their actions incrementally, step by step. This is the case of many applications in robotics, e.g., in SLAM (Stachniss, Leonard, and Thrun 2016), where the robot tries to build a map of the environment by exploration, or in the Robocup Rescue (Kitano and Tadokoro 2001), where the robot needs to explore the environment to perform a rescue task. Second, previous work on learning action models does not deal with the problem of generating informative plan traces. As stated in the conclusions of (Aineto, Jiménez Celorrio, and Onaindia 2019), generating informative plan traces for learning planning action models is still an open issue. Indeed, if the available set of plan traces does not contain informative examples, there is little chance to learn all action preconditions, since some preconditions can be only discovered by specific plans that can unlikely be generated randomly (Fern, Yoon, and Givan 2004).

In this paper, we propose a new approach that does not suffer these drawbacks, focusing on the case of learning STRIPS action schema expressed in PDDL, and under the assumption of full observability of the states reached by the agent. We propose an algorithm, called OLAM algorithm (Online Learning of Action Models), for learning action models online, incrementally during the execution of plans. A key aspect of OLAM is that it combines and interleaves the activity of learning action preconditions and effects with an exploration phase that selects which plan to execute. In this way, OLAM generates plan traces to reach certain goal states, decided online, which are useful for the learning task.

Beyond proving termination, we analyse our algorithm to show some important theoretical properties that are defined according to the state transitions of the models learned by the algorithm. In particular we prove that OLAM is correct, i.e., it learns action models which generate only the state transitions generated by the planning domain modelling the true environment where the agent acts. Moreover OLAM is "integral", i.e., it learns action models that generate all the transitions of the true environment with respect to the states that can be reached by the algorithm.

We also provide substantial empirical evidence of the good learning performance of OLAM using a large set of benchmarks from the International Planning Competitions (IPCs). Finally we experimentally compare OLAM with a recent and state-of-the-art method for learning action models

offline, showing that the online learning can be much more effective.

Related work

Recent offline approaches address the problem of model learning with different assumptions on the observability of states and actions, see, e.g., (Amir and Chang 2008; Bonet and Geffner 2020; Cresswell, McCluskey, and West 2013; Mourão et al. 2012; Newton et al. 2007; Yang, Wu, and Jang 2007; Zhuo et al. 2010; Zhuo and Kambhampati 2013). A prominent system among these is Fama (Aineto, Jiménez Celorrio, and Onaindia 2019), which learns action models offline from examples by transforming the learning task into a classical planning task. It works with different kinds of inputs, from a set of plans to just a pair of initial and final states, without intermediate actions or states. Moreover, it accepts in input partially specified action models.

On the one hand, the aforementioned approaches to offline learning can deal with partial observability of states and actions, and some of them even with noisy states and noisy actions. OLAM requires instead full observability of states, it does not deal with noisy sensors, and actions are decided by OLAM itself. On the other hand, differently from OLAM, all these approaches are offline, require in input plan traces that in some cases might be not available, and hence do not deal with the issue of selecting informative plan traces.

Since the seminal work on online learning of operators (Gil 1994b,a; Wang 1996), and the first approaches to learning action models by integrating learning, planning, and execution (García-Martínez and Borrajo 2000), some recent approaches have addressed the problem of online and incremental learning of action models. Walsh and Littman (2008) propose an approach to online learn action models which can be used in web-service planning problems. Their approach requires the use of an external "teacher" providing plan traces on demand. 3SG (Certicky 2014) is an online algorithm that learns probabilistic action models with conditional effects and deals with action failures, sensory noise, and incomplete information. Xu and Laird (2010) describes an instance-based online method for learning action models in relational domains. The work is extended to deal with both discrete and continuous action models (Xu and Laird 2011, 2013). Rodrigues et al. (2010; 2010) propose a technique based on relational reinforcement learning to learn deterministic action models, and Rodrigues et al. (2011) extend the approach to deal with non-deterministic actions. These approaches are based on important technical differences with respect to our work. Most important, the main conceptual and practical difference is that all these approaches assume that the action to be executed is randomly selected or given in input, and therefore do not deal with the problem of guiding the exploration phase towards informative states, a key and promising feature of OLAM. The work by Lamanna et al. (2021) proposes an online method to learn planning domains by mapping continuous observations to deterministic state transition systems. It uses a given PDDL planning domain and a classical planner to heuristically explore the state space towards the problem goal. However, it focuses on learning the final state machine of the planning domain rather than PDDL action models like OLAM.

Our approach shares some similarities with the work on planning by reinforcement learning (RL) (Sutton and Barto 1998). However, RL focuses on learning policies rather than PDDL action models. Moreover, most often, in RL, actions are represented as (probabilistic) state transitions, rather than with symbolic action models.

Problem

Let $\mathcal{P}$ be a set of predicates with associated arity, of a first order language, and $\mathcal{O}$ be a finite set of operator names with associated arity. Predicates and operators of arity n are called n-ary predicates and n-ary operators. Given an n-tuple $\boldsymbol{x} = \langle x_1, \ldots, x_n \rangle$ of distinct symbols (constants or variables), let $\mathcal{P}(\boldsymbol{x})$ be the set of atomic formulas $p(x_{i_1}, \ldots, x_{i_m})$ obtained by applying the m-ary predicate $p \in \mathcal{P}$ to any m-tuple of symbols $\langle x_{i_1}, \ldots, x_{i_m} \rangle$ in $\boldsymbol{x}$ (with $1 \leq i_1, \ldots, i_m \leq n$). For instance, if $\mathcal{P}$ contains the single binary predicate on, and $\boldsymbol{x} = \langle x_1, x_2, x_3 \rangle$. Then, $\mathcal{P}(\boldsymbol{x}) = \{\texttt{on}(x_i, x_j) \,|\, 1 \leq i, j \leq 3\}$.

Definition 1 (Action schema)**.** *An* action schema *for an n-ary operator name $op \in \mathcal{O}$ on the set of predicates $\mathcal{P}$ is a tuple $\big\langle \mathsf{par}(op), \mathsf{pre}(op), \mathsf{eff}^+(op), \mathsf{eff}^-(op) \big\rangle$, where $\mathsf{par}(op)$ is a tuple of variables, $\mathsf{pre}(op)$, $\mathsf{eff}^+(op)$, and $\mathsf{eff}^-(op)$ are three sets of atoms on $\mathcal{P}(\mathsf{par}(op))$.*

Essentially, $\mathsf{pre}(op)$, $\mathsf{eff}^+(op)$, and $\mathsf{eff}^-(op)$ represent the preconditions, positive, and negative effects of operator op. Without loss of generality, we assume that operators have no negative precondition. We also assume that the description of the effects is consistent, i.e., $\mathsf{eff}^+(op) \cap \mathsf{eff}^-(op) = \emptyset$.

Definition 2 (Ground action)**.** *The* ground action $a = op(c_1, \ldots, c_n)$ *of an n-ary operator name $op \in \mathcal{O}$ w.r.t. the constants $c_1, \ldots, c_n$ is the triple $\langle \mathsf{pre}(a), \mathsf{eff}^+(a), \mathsf{eff}^-(a) \rangle$, where $\mathsf{pre}(a)$ (resp. $\mathsf{eff}^+(a)$, $\mathsf{eff}^-(a)$) is obtained by replacing the i-th parameter of $\mathsf{par}(op)$ in $\mathsf{pre}(op)$ (resp. $\mathsf{eff}^+(op)$, $\mathsf{eff}^-(op)$) with c_i.*

We use the term *lifted*, as the opposite of *grounded*, to refer to expressions and actions where constants have been replaced with parameters.

Definition 3 (Planning domain)**.** *A planning domain $\mathcal{M}$ is a triple $\langle \mathcal{P}, \mathcal{O}, \mathcal{H} \rangle$ where $\mathcal{P}$ is a set of predicates, $\mathcal{O}$ is a set of operator names with their arity and, for every $op \in \mathcal{O}$, $\mathcal{H}$ is a function mapping an operator name op into an action schema.*

Definition 4 (Finite-State Machine of a planning domain)**.** *The* Finite-State Machine (FSM) *of a planning domain $\mathcal{M} = \langle P, \mathcal{O}, \mathcal{H} \rangle$ for the set $\mathcal{C}$ of constants is the triple $\mathcal{M}(\mathcal{C}) = \langle \mathcal{S}, \mathcal{A}, \delta \rangle$ where $\mathcal{S} = 2^{\mathcal{P}(\mathcal{C})}$ is the set of all possible subsets of facts; $\mathcal{A}$ is the set of all possible ground actions of each n-ary operator name in $\mathcal{O}$ on any n-tuple of constants in $\mathcal{C}$; $\delta \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is a transition relation such that $(s, a, s') \in \delta$ if $\mathsf{pre}(a) \subseteq s$ and $s' = s \cup \mathsf{eff}^+(a) \setminus \mathsf{eff}^-(a)$.*

A *plan* π in $\mathcal{M}(\mathcal{C})$ is a finite sequence of actions. A state $s_n \in S$ is *reachable* from a state $s_0 \in S$ in $\mathcal{M}(\mathcal{C})$ if there

is a plan $\pi = \langle a_1, \ldots, a_n \rangle$ such that $(s_{i-1}, a_i, s_i) \in \delta$ for $i = 1 \ldots n$.

Assuming that the sets $\mathcal{P}$, $\mathcal{O}$ and $\mathcal{C}$ are known by the agent, its task is to *learn a planning domain by executing the actions available in $\mathcal{O}$ over constants in $\mathcal{C}$, observing, and determining what are their preconditions and effects on the environment described in terms of the properties in $\mathcal{P}$. In formal terms, the agent has to build an action model $\mathcal{M} = \langle \mathcal{P}, \mathcal{O}, \mathcal{H} \rangle$, i.e., the preconditions and effects of every action schema in the domain of $\mathcal{H}$.* We assume that the dynamics of the environment where the agent acts, which is unknown by the agent, is fully described by the finite state machine $\mathcal{M}'(\mathcal{C})$, where $\mathcal{M}' = \langle \mathcal{P}, \mathcal{O}, \mathcal{H}' \rangle$ is an action model called *Ground-Truth Model* (GTM).

The following definitions state the notions of correctness and integrity for the learned planning domain $\mathcal{M} = \langle P, \mathcal{O}, \mathcal{H} \rangle$ w.r.t. the GTM.

Definition 5 (Correctness)**.** *Let $\mathcal{M}$ and $\mathcal{M}'$ be two action models and $\mathcal{M}(\mathcal{C}) = \langle \mathcal{S}, \mathcal{A}, \delta \rangle$ and $\mathcal{M}'(\mathcal{C}) = \langle \mathcal{S}, \mathcal{A}, \delta' \rangle$ be their FSMs with respect to a set of constants $\mathcal{C}$. We say that*

1. *$\mathcal{M}(\mathcal{C})$* correctly approximates *$\mathcal{M}'(\mathcal{C})$ from a state $s_0 \in \mathcal{S}$ if, for every state s_n reachable from s_0 in $\mathcal{M}(\mathcal{C})$, $\langle s_n, a, s \rangle \in \delta$ implies $\langle s_n, a, s' \rangle \in \delta'$ for some $s' \supseteq s$.*
2. *$\mathcal{M}(\mathcal{C})$* correctly approximates *$\mathcal{M}'(\mathcal{C})$ if $\mathcal{M}(\mathcal{C})$ correctly approximates $\mathcal{M}'(\mathcal{C})$ from every state in $\mathcal{S}$;*
3. *$\mathcal{M}$* correctly approximates *$\mathcal{M}'$ if $\mathcal{M}(\mathcal{C})$ correctly approximates $\mathcal{M}'(\mathcal{C})$ for every set of constants $\mathcal{C}$.*

A plan is *valid* when the actions in the plan are "executable" and the plan achieves a given set of (positive) goals. Therefore, when the learned model correctly approximates the GTM, any valid plan computed by using the learned model is also valid for the GTM.

Definition 6 (Integrity)**.** *Let $\mathcal{M}$ and $\mathcal{M}'$ be two action models and $\mathcal{M}(\mathcal{C}) = \langle \mathcal{S}, \mathcal{A}, \delta \rangle$ and $\mathcal{M}'(\mathcal{C}) = \langle \mathcal{S}, \mathcal{A}, \delta' \rangle$ be their FSMs with respect to a set of constants $\mathcal{C}$. We say that*

1. *$\mathcal{M}(\mathcal{C})$* integrally approximates *$\mathcal{M}'(\mathcal{C})$ from a state $s_0 \in \mathcal{S}$ if, for every state s_n reachable from s_0 in $\mathcal{M}(\mathcal{C})$, $\langle s_n, a, s' \rangle \in \delta'$ implies $\langle s_n, a, s \rangle \in \delta$ for some $s \supseteq s'$;*
2. *$\mathcal{M}(\mathcal{C})$* integrally approximates *$\mathcal{M}'(\mathcal{C})$ if $\mathcal{M}(\mathcal{C})$ integrally approximates $\mathcal{M}'(\mathcal{C})$ from every state in $\mathcal{S}$;*
3. *$\mathcal{M}$* integrally approximates *$\mathcal{M}'$ if $\mathcal{M}(\mathcal{C})$ integrally approximates $\mathcal{M}'(\mathcal{C})$ for every set of constants $\mathcal{C}$.*

Therefore, when the learned model integrally approximates the GTM, any valid plan for the GTM is also a valid plan for the learned model.

Learning algorithm

In the proposed approach, the agent constructs and executes informative plan traces for learning the planning domain. Algorithm 1 shows the pseudocode of the OLAM (*Online Learning of Action Models*). The input of the algorithm are the same sets of predicates and operator names (with their associated arity) of the GTM, and a set $\mathcal{C}$ of constants representing the objects of the environment explored by the agent. OLAM produces in the output two planning domains $\mathcal{M}$ and $\mathcal{M}^-_?$. The former is such that $\mathcal{M}(\mathcal{C})$ correctly and integrally approximates $\mathcal{M}'(\mathcal{C})$ from the state of the environment when OLAM terminates. The latter correctly approximates $\mathcal{M}'$.

We adopt the following notations. Let $\boldsymbol{x} = \langle x_1, \ldots, x_n \rangle$ and $\boldsymbol{c} = \langle c_1, \ldots, c_n \rangle$ two n-tuple of distinct parameters and constants. If p is an m-ary predicate, $p(\boldsymbol{x})$ denotes an atom $p(x_{i_1}, \ldots, x_{i_m})$ for some m-tuple of indexes $1 \leq i_1, \ldots, i_m \leq n$; and $p(\boldsymbol{c})$ the atom obtained by replacing x_i with c_i in $p(\boldsymbol{x})$. In the following the indexing will be left implicit. OLAM incrementally builds the following sets:

1. $\mathsf{pre}(op)$, which contains the preconditions of the operator op; it is initialized to the whole set of lifted atoms (line 2); an atom $p(\boldsymbol{x})$ is removed from $\mathsf{pre}(op)$ whenever an instance $op(\boldsymbol{c})$ of op is executed successfully in a state s and $p(\boldsymbol{c}) \notin s$ (line 19).
2. $\mathsf{eff}^+_!(op)$ and $\mathsf{eff}^-_!(op)$, which contains the set of lifted positive and negative effects of op learned by the agent; they are initially empty (line 3); a lifted atom $p(\boldsymbol{x})$ is added to $\mathsf{eff}^+_!(op)$ (resp. $\mathsf{eff}^-_!(op)$) if the execution of an instance $op(\boldsymbol{c})$ of op in state s makes $p(\boldsymbol{c})$ become true (resp. false) in the resulting state (lines 20-21).
3. $\mathsf{eff}^+_?(op)$ and $\mathsf{eff}^-_?(op)$, which are sets of lifted atoms that could become part of the positive or negative effects of op; they are initialized to the entire set of lifted atoms (line 2); a lifted atom $p(\boldsymbol{x})$ is removed from $\mathsf{eff}^+_?(op)$ (resp. $\mathsf{eff}^-_?(op)$) if $p(\boldsymbol{x})$ is discovered to be a positive or negative effect or if the atom $p(\boldsymbol{c})$ is false (resp. true) in a state s and remains false (resp. true) after executing successfully an instance $op(\boldsymbol{c})$ of op in s (lines 22-23).
4. $\mathsf{pre}_\perp(op)$, which is a set of sets of lifted preconditions for op such that in every non empty set in $\mathsf{pre}_\perp(op)$ there is at least one precondition of op; $\mathsf{pre}_\perp(op)$ is initialized to a set including only the empty set (line 4); $\mathsf{pre}_\perp(op)$ is augmented by the set formed by any lifted fact $p(\boldsymbol{x})$ such that $p(\boldsymbol{c})$ is false in a state s, if the execution of an instance $op(\boldsymbol{c})$ of op fails in s (line 26).
5. $\mathsf{eff}^+_{!?}(op)$ and $\mathsf{eff}^-_{!?}(op)$, which are derived sets denoting $\mathsf{eff}^+_!(op) \cup \mathsf{eff}^+_?(op)$ and $\mathsf{eff}^-_!(op) \cup \mathsf{eff}^-_?(op)$.

At each iteration of the external loop (lines 7–31), the agent selects a state s' and a ground action $op'(\boldsymbol{c}')$. s' is reachable from the current state with the model $\mathcal{M}$ learned so far; the ground action $op'(\boldsymbol{c}')$ is such that its execution in s' could provide to the agent some additional information about the preconditions, the positive, or the negative effects of op'. This condition is formalised by (2)–(4). In particular, if condition (2) holds, the preconditions of op' could be refined by executing $op'(\boldsymbol{c}')$ in s', because s' does not contain all the preconditions of $op'(\boldsymbol{c}')$. Indeed if $op'(\boldsymbol{c}')$ will succeed, then the preconditions which are false in s' can be eliminated. If condition (3) (resp. (4)) holds, some positive (resp. negative) effects of op' could be learned, because $op'(\boldsymbol{c}')$ is executable in s' and s' does not contain all the facts in $\mathsf{eff}^+_?(op'(\boldsymbol{c}'))$ (resp. contains at least a fact in $\mathsf{eff}^-_?(op'(\boldsymbol{c}'))$). Indeed, the facts in $\mathsf{eff}^+_?(op'(\boldsymbol{c}'))$ but not in s' which become true can be added to the positive effects.

Similarly, the facts that are in $\mathsf{eff}^-_?(op'(\boldsymbol{c}'))$ and in s' which become false can be added to the negative effects. The selection of such a state s' and action $op'(\boldsymbol{c}')$ is done by constructing a plan from the current state s to a state s' which satisfies conditions (2)–(4) for an $op'(\boldsymbol{c}')$ (line 8). If there is more than one action $op'(\boldsymbol{c}')$ that satisfies conditions (2)–(4) in s', one of them is randomly selected. The choice of s', $op'(c')$ and the associated plan is obtained by invoking PLAN with the following goal:

$$G = \bigvee_{\substack{op(\boldsymbol{c})\in\mathcal{A} \\ P^+P^-E^+E^- \text{ satisfy } (i\text{–}vi)}} \left(\bigwedge_{p(\boldsymbol{c})\in P^+\cup E^-} p(\boldsymbol{c}) \wedge \bigwedge_{p(\boldsymbol{c})\in P^-\cup E^+} \neg p(\boldsymbol{c}) \right) \tag{1}$$

$$(i)\ P^-\cup E^+\cup E^-\neq\emptyset, \quad (ii)\ P^+\cap P^-=\emptyset,$$
$$(iii)\ P^+\cup P^- = \mathsf{pre}(op(\boldsymbol{c})), (iv)\ P^-\notin \mathsf{pre}_\perp(op(\boldsymbol{c}))\setminus\{\emptyset\},$$
$$(v)\ E^+\subseteq \mathsf{eff}^+_?(op(\boldsymbol{c})), \quad (vi)\ E^-\subseteq \mathsf{eff}^-_?(op(\boldsymbol{c})).$$

Each disjunct in (1) describes a set of states from which the agent can potentially learn something by executing $op(\boldsymbol{c})$. P^+ and P^- partition the preconditions of $op(\boldsymbol{c})$ so that the atoms in P^+ are true in s', the atoms in P^- are false in s', and set P^- has not already been checked to be necessary for successfully executing $op(\boldsymbol{c})$. E^+ is a subset of possible positive effects of $op(\boldsymbol{c})$ which are false in s' and can become true by executing $op(\boldsymbol{c})$; similarly for E^-. Notice that for every state s' that contains P^+ and E^- and does not contain P^- and E^+, and every action $op'(\boldsymbol{c}')$ such that (iv) and (v) and (vi) hold, when condition (2) is satisfied by s' and $op'(\boldsymbol{c}')$, P^- is not empty; when condition (3) is satisfied by s' and $op'(\boldsymbol{c}')$, E^+ is not empty; finally, when condition (4) is satisfied by s' and $op'(\boldsymbol{c}')$, E^- is not empty.

In the internal loop (lines 9–30), OLAM executes π and if it manages to successfully complete the execution of π (i.e., $\pi=\langle\rangle$, line 10) the ground action $op'(\boldsymbol{c}')$ will be executed in the environment where the agent acts (line 17). The dynamics of such an environment is unknown by the agent, and it determines the result returned by call EXECUTE($op(\boldsymbol{c})$). When a ground action $op(\boldsymbol{c})$ is successfully executed, OLAM observes the state of the environment s_{next} resulting from the execution (line 18), and updates sets $\mathsf{pre}(op)$, $\mathsf{eff}^{+/-}_!(op)$ and $\mathsf{eff}^{+/-}_?(op)$ according to the criteria defined above (lines 19–23). If the $op(\boldsymbol{c})$ execution fails in the environment, $\mathsf{pre}_\perp(op)$ is extended as described above, and π is reset to nil since its execution deviates from the expected trajectory computed according to the domain $\mathcal{M}$ learned so far (lines 26-27).

Termination, correctness and integrity

Given an n-ary operator, we assume that it can be grounded only with n *different* constants. This assumption can be done without loss of generality, at the price of introducing additional operators with only one parameter in place of the set of parameters that can be grounded with the same constant. We also suppose that OLAM is run in the environment $\mathcal{M}'(\mathcal{C})$ where $\mathcal{C}$ is a set of at least $\max_{op\in\mathcal{O}}|\mathsf{par}(op)|$ constants.

Algorithm 1 OLAM

Require: $\mathcal{M}=\langle\mathcal{P},\mathcal{O},\{\mathsf{par}(op),\emptyset,\emptyset\}_{op\in\mathcal{O}}\rangle, \mathcal{C}$
1: $s\leftarrow$ OBSERVE()
2: $\forall op\in\mathcal{O}, \mathsf{eff}^-_?(op)\leftarrow\mathsf{eff}^+_?(op)\leftarrow\mathsf{pre}(op)\leftarrow\mathcal{P}(\mathsf{par}(op))$
3: $\forall op\in\mathcal{O}, \mathsf{eff}^-_!(op)\leftarrow\mathsf{eff}^+_!(op)\leftarrow\emptyset$
4: $\forall op\in\mathcal{O}, \mathsf{pre}_\perp(op)\leftarrow\{\emptyset\}$
5: $\mathcal{M}\leftarrow\langle\mathcal{P},\mathcal{O},\{\mathsf{par}(op),\mathsf{pre}(op),\mathsf{eff}^+_!(op),\mathsf{eff}^-_!\}_{op\in\mathcal{O}}\rangle$
6: $\pi\leftarrow nil$
7: **while** $\exists s', op'(\boldsymbol{c}')$ such that s' is reachable from s by $\mathcal{M}(\mathcal{C})$ and (2) $\vee$ (3) $\vee$ (4) holds **do**
8: $\quad\pi\leftarrow$ PLAN($\mathcal{M}(\mathcal{C}), s, s'$)
9: $\quad$**while** $\pi\neq nil$ **do**
10: $\quad\quad$**if** $\pi\neq\langle\rangle$ **then**
11: $\quad\quad\quad op(\boldsymbol{c})\leftarrow$ POP(π)
12: $\quad\quad$**else**
13: $\quad\quad\quad op(\boldsymbol{c})\leftarrow op'(\boldsymbol{c}')$
14: $\quad\quad\quad\pi\leftarrow nil$
15: $\quad\quad$**end if**
16: $\quad\quad\boldsymbol{x}\leftarrow\mathsf{par}(op)$
17: $\quad\quad$**if** EXECUTE($op(\boldsymbol{c})$) does not fail **then**
18: $\quad\quad\quad s_{next}\leftarrow$ OBSERVE()
19: $\quad\quad\quad\mathsf{pre}(op)\leftarrow\{p(\boldsymbol{x})\in\mathsf{pre}(op)\mid p(\boldsymbol{c})\in s\}$
20: $\quad\quad\quad\mathsf{eff}^+_!(op)\leftarrow\mathsf{eff}^+_!(op)\cup\{p(\boldsymbol{x})\mid p(\boldsymbol{c})\in s_{next}\setminus s\}$
21: $\quad\quad\quad\mathsf{eff}^-_!(op)\leftarrow\mathsf{eff}^-_!(op)\cup\{p(\boldsymbol{x})\mid p(\boldsymbol{c})\in s\setminus s_{next}\}$
22: $\quad\quad\quad\mathsf{eff}^+_?(op)\leftarrow\mathsf{eff}^+_?(op)\setminus\{p(\boldsymbol{x})\mid p(\boldsymbol{c})\notin s\cap s_{next}\}$
23: $\quad\quad\quad\mathsf{eff}^-_?(op)\leftarrow\mathsf{eff}^-_?(op)\setminus\{p(\boldsymbol{x})\mid p(\boldsymbol{c})\in s\cup s_{next}\}$
24: $\quad\quad\quad s\leftarrow s_{next}$
25: $\quad\quad$**else**
26: $\quad\quad\quad\mathsf{pre}_\perp(op)\leftarrow\mathsf{pre}_\perp(op)\cup\{\{p(\boldsymbol{x})\in\mathsf{pre}(op)\mid p(\boldsymbol{c})\notin s\}\}$
27: $\quad\quad\quad\pi\leftarrow nil$
28: $\quad\quad$**end if**
29: $\quad\quad\mathcal{M}\leftarrow\langle\mathcal{P},\mathcal{O},\{\mathsf{par}(op),\mathsf{pre}(op),\mathsf{eff}^+_!(op),\mathsf{eff}^-_!\}_{op\in\mathcal{O}}\rangle$
30: $\quad$**end while**
31: **end while**
32: $\mathcal{M}^-_?\leftarrow\langle\mathcal{P},\mathcal{O},\{\mathsf{par}(op),\mathsf{pre}(op),\mathsf{eff}^+_!(op),\mathsf{eff}^-_{!?}\}_{op\in\mathcal{O}}\rangle$
33: **return** $\mathcal{M}, \mathcal{M}^-_?$

Conditions in line 7:

$$\mathsf{pre}(op'(\boldsymbol{c}'))\setminus s'\notin\mathsf{pre}_\perp(op'(\boldsymbol{c}')) \tag{2}$$
$$\mathsf{pre}(op'(\boldsymbol{c}'))\subseteq s' \text{ and } \mathsf{eff}^+_?(op'(\boldsymbol{c}'))\not\subseteq s' \tag{3}$$
$$\mathsf{pre}(op'(\boldsymbol{c}'))\subseteq s' \text{ and } \mathsf{eff}^-_?(op'(\boldsymbol{c}'))\cap s'\neq\emptyset \tag{4}$$

In the following, the transitions relations of $\mathcal{M}$, $\mathcal{M}'$, and $\mathcal{M}^-_?$ are denoted by δ, δ', and $\delta^-_?$. Moreover, the sets of preconditions and positive/negative effects of any operator op of $\mathcal{M}'$ are denoted by $\mathsf{pre}'(op)$ and $\mathsf{eff}'^{+/-}(op)$, respectively.

Lemma 1. *For every n-ary operator op with parameters $\mathsf{par}(op)=\boldsymbol{x}=(x_1,\ldots,x_n)$, every m-ary predicate p, and every n-tuple of distinct constants $\mathbf{c}=(c_1,\ldots,c_n)$ in $\mathcal{C}$:*

1. $p(\boldsymbol{c}')\in\mathsf{pre}(op(\boldsymbol{c}))$ *iff* $p(\boldsymbol{x}')\in\mathsf{pre}(op)$
2. $p(\boldsymbol{c}')\in\mathsf{eff}^+(op(\boldsymbol{c}))$ *iff* $p(\boldsymbol{x}')\in\mathsf{eff}^+(op)$
3. $p(\boldsymbol{c}')\in\mathsf{eff}^-(op(\boldsymbol{c}))$ *iff* $p(\boldsymbol{x}')\in\mathsf{eff}^-(op)$

where $\boldsymbol{c}'=(c_{i_1},\ldots,c_{i_m})$ and $\boldsymbol{x}'=(x_{i_1},\ldots,x_{i_m})$ with $1\leq i_j\leq n$ for $1\leq j\leq m$.

Proof. Let us consider the case 1 (similar proofs can be

derived for cases 2 and 3). Since $\mathsf{pre}(op)$ cannot contain constants, the only way to obtain $p(\boldsymbol{c}')$ in $\mathsf{pre}(op(\boldsymbol{c}))$ is by grounding some precondition $p(\boldsymbol{x}') \in \mathsf{pre}(op)$ with $\boldsymbol{c}'$. Since we require that every parameter in $\mathsf{par}(op)$ is instantiated with a different constant, the only way to obtain $p(c_{i_1}, \ldots, c_{i_m})$ is when $\boldsymbol{x}' = (x_{i_1}, \ldots, x_{i_m})$. The opposite direction derives by the fact that if $p(\boldsymbol{x}') \in \mathsf{pre}(op)$, by grounding op's parameters in $p(\boldsymbol{x}')$ with $\boldsymbol{c}'$, we obtain $p(\boldsymbol{c}') \in \mathsf{pre}(op(\boldsymbol{c}))$. □

Lemma 2. *At every execution step of OLAM* $\mathsf{pre}(op) \supseteq \mathsf{pre}'(op)$.

Proof. In OLAM, $\mathsf{pre}(op)$ is initialized by $\mathcal{P}(\mathsf{par}(op))$, i.e., all the possible preconditions on the parameters of op (line 2). Then, a precondition $p(\boldsymbol{x})$ is removed from $\mathsf{pre}(op)$ when an action $op(\boldsymbol{c})$ is executed with success in s and $p(\boldsymbol{c}) \notin s$. (line 19). This implies that $p(\boldsymbol{c}) \notin \mathsf{pre}'(op(\boldsymbol{c}))$. By lemma 1 we have that $p(\boldsymbol{x}) \notin \mathsf{pre}'(op)$. This guarantees that at every execution step of the algorithm $\mathsf{pre}(op) \supseteq \mathsf{pre}'(op)$. □

Lemma 3. *At every execution step of OLAM, if* $\mathsf{pre}(op(\boldsymbol{c})) \subseteq$ OBSERVE(), *then* EXECUTE($op(\boldsymbol{c})$) *does not fail.*

Proof. Let $s =$ OBSERVE() be the result of the observation of the environment at some execution step of OLAM, i.e., the current state according to $\mathcal{M}'(\mathcal{C})$. By Lemmas 1-2, if $\mathsf{pre}(op(\boldsymbol{c})) \subseteq s$ then $\mathsf{pre}'(op(\boldsymbol{c})) \subseteq s$, which guarantees that action $op(\boldsymbol{c})$ can be executed with success from the current state according to $\mathcal{M}'(\mathcal{C})$. □

Theorem 1 (Termination)**.** *Algorithm OLAM terminates.*

Proof. First of all notice that for every operator op the following properties hold:

- The size of $\mathcal{P}(\mathsf{par}(op))$ and $2^{\mathcal{P}(\mathsf{par}(op))}$ are finite and therefore $\mathsf{pre}(op)$, $\mathsf{eff}^+_?(op)$, $\mathsf{eff}^-_?(op)$ are initialized to finite sets, $\mathsf{eff}^+_!(op)$, $\mathsf{eff}^-_!(op)$ cannot be larger than $\mathcal{P}(\mathsf{par}(op))$, and $\mathsf{pre}_\perp(op)$ cannot be larger then the size of $2^{\mathcal{P}(\mathsf{par}(op))}$.
- The internal loop (lines 9–30) always terminates because at every iteration either the size of the plan reduces of 1 unit, or the plan is set to nil, and if the size is 0 the plan is set to nil.

Given the above points, to show termination, we have to prove that at every iteration of the external loop (7–31) one of the following facts holds for some operator op:

(a) the size of $\mathsf{pre}(op)$, $\mathsf{eff}^+_?(op)$, or $\mathsf{eff}^-_?(op)$ is reduced;

(b) the size of $\mathsf{pre}_\perp(op)$, $\mathsf{eff}^+_!(op)$, or $\mathsf{eff}^-_!(op)$ is increased.

At each iteration of the external loop (lines 7–31) OLAM selects an action $op'(\boldsymbol{c}')$ and a state s' reachable from the current state s with $\mathcal{M}$ that satisfy condition (2), (3) or (4). It produces a plan $\pi = (a_1, \ldots, a_k)$ from the current state to s', and it executes the plan in the internal loop (lines 9–30). We consider separately the case where (i) π is successfully executed and the state s' is achieved, and (ii) the execution of π fails or the reached state is different from s'.

(i) *π successfully achieves s':* after k iterations of the internal loop, plan π becomes empty, and the condition at line 10 becomes true. Then, $op'(\boldsymbol{c}')$ is executed in s' (line 11). If $op'(\boldsymbol{c}')$ is executed successfully, then since $op'(\boldsymbol{c}')$ and s' satisfies at least one of the three conditions (2), (3) and (4), the following applies. If (2) holds then $\mathsf{pre}(op(\boldsymbol{c})) \setminus s' \neq \emptyset$ and therefore $\mathsf{pre}(op')$ is reduced (line 19); if (3) holds then $\mathsf{eff}^+_?(op)$ is reduced (line 22); if (4) holds then $\mathsf{eff}^-_?(op)$ is reduced (line 23). If $op'(\boldsymbol{c}')$ fails in the state s' (line 25), then by Lemma 3 it means that $\mathsf{pre}(op'(\boldsymbol{c}')) \not\subseteq s'$, and therefore conditions (3) and (4) are false, which implies that condition (2) is true. This guarantees that at line 26 $\mathsf{pre}_\perp(op)$ is extended.

(ii) *π fails to achieve s':* Since plan π is computed according to $\mathcal{M}$ and, by Lemmas 1-2, for any action $op(\boldsymbol{c})$ the set of preconditions of $op(\boldsymbol{c})$ in $\mathcal{M}(\mathcal{C})$ contains the preconditions of $op(\boldsymbol{c})$ in $\mathcal{M}'(\mathcal{C})$, then the failure of π implies that after $j \leq k$ iterations of the internal loop, the observed state (computed by executing $a_1, \ldots, a_j$ from s in $\mathcal{M}'(\mathcal{C})$) is different from the state computed by executing $a_1, \ldots, a_j$ from s in $\mathcal{M}$. Let i be the smallest of such a j, and $a_i = op_i(\boldsymbol{c}_i)$. Then $(s_{i-1}, op_i(\boldsymbol{c}_i), s_i) \in \delta$ and $(s_{i-1}, op_i(\boldsymbol{c}_i), s'_i) \in \delta'$ with $s'_i \neq s_i$. Since $a_{i-1} = op_{i-1}(\boldsymbol{c}_{i-1})$ modifies only the atoms containing the constants $\boldsymbol{c}_{i-1}$ contained in s_i and s'_i differ on some $p(\boldsymbol{c}_{i-1})$. If $p(\boldsymbol{c}_{i-1}) \in s'_i \setminus s_i$ then, at the $i-1$-th iteration of the internal loop, $\mathsf{eff}^+_!(op_{i-1})$ is extended (line 20); if $p(\boldsymbol{c}_{i-1}) \in s_i \setminus s'_i$, then $\mathsf{eff}^-_!(op_{i-1})$ is extended (line 21).

□

Lemma 4. *At every execution step of OLAM,* $\mathsf{eff}^{+/-}_!(op) \subseteq \mathsf{eff}'^{+/-}(op)$.

Proof. In OLAM, $\mathsf{eff}^{+/-}_!(op)$ are initialized by the empty set (line 3), and therefore $\mathsf{eff}^{+/-}_!(op) \subseteq \mathsf{eff}'^{+/-}(op)$ initially holds. Suppose that at some point $p(\boldsymbol{x}) \in \mathsf{eff}^+_!(op)$. Then, there exists a state s and an action $op(\boldsymbol{c})$ such that the execution of $op(\boldsymbol{c})$ from s adds $p(\boldsymbol{x})$ to $\mathsf{eff}^+_!(op)$, which implies $p(\boldsymbol{c}) \notin s$ and $p(\boldsymbol{c}) \in s_{next}$ (line 20), where s_{next} is the state resulting from the execution of $op(\boldsymbol{c})$ in $\mathcal{M}'(\mathcal{C})$. Since $s_{next} \subseteq s \cup \mathsf{eff}'^{+}(op(\boldsymbol{c}))$, then $p(\boldsymbol{c}) \in \mathsf{eff}'^{+}(op(\boldsymbol{c}))$. By Lemma 1, we have that $p(\boldsymbol{x}) \in \mathsf{eff}'^{+}(op)$. Similar proof can be done to show that $\mathsf{eff}^-_!(op) \subseteq \mathsf{eff}'^{-}(op)$. □

Lemma 5. *At every execution step of OLAM,* $\mathsf{eff}^{+/-}_{!?}(op) \supseteq \mathsf{eff}'^{+/-}(op)$.

Proof. In OLAM, $\mathsf{eff}^-_?(op)$ is initialized by $\mathcal{P}(\mathsf{par}(op))$ (line 2), and therefore $\mathsf{eff}^-_{!?}(op) \supseteq \mathsf{eff}^-_? \supseteq \mathsf{eff}'^{-}(op)$ initially holds. Suppose that at some point $p(\boldsymbol{x}) \notin \mathsf{eff}^-_{!?}(op)$. This means that there is a state s and an action $op(\boldsymbol{c})$ such that the execution of $op(\boldsymbol{c})$ from s removes $p(\boldsymbol{x})$ from $\mathsf{eff}^-_?(op)$, which implies that $p(\boldsymbol{c}) \in s \cup s_{next}$ (line 21), where $s_{next} =$

$s \cup \mathsf{eff}'^{+}(op(\boldsymbol{c})) \setminus \mathsf{eff}'^{-}(op(\boldsymbol{c}))$ is the state resulting from the execution of $op(\boldsymbol{c})$ in $\mathcal{M}'(\mathcal{C})$. If $p(\boldsymbol{c}) \notin s_{next}$ then $p(\boldsymbol{c}) \in s$ and this implies that $p(\boldsymbol{x})$ is added to $\mathsf{eff}^{-}_{!}(op)$, which contradicts the fact that $p(\boldsymbol{x}) \notin \mathsf{eff}^{-}_{!?}(op)$; therefore we have that $p(\boldsymbol{c}) \in s_{next}$, which implies that $p(\boldsymbol{x}) \notin \mathsf{eff}'^{-}(op)$, as required.

In OLAM, $\mathsf{eff}^{+}_{?}(op)$ is initialized by $\mathcal{P}(\mathsf{par}(op))$ (line 2), and therefore $\mathsf{eff}^{+}_{!?}(op) \supseteq \mathsf{eff}^{+}_{?} \supseteq \mathsf{eff}'^{+}(op)$ initially holds. Suppose that at some point $p(\boldsymbol{x}) \notin \mathsf{eff}^{+}_{!?}(op)$. This means that there is a state s and an action $op(\boldsymbol{c})$ such that the execution of $op(\boldsymbol{c})$ from s removes $p(\boldsymbol{x})$ from $\mathsf{eff}^{+}_{?}(op)$, which implies that $p(\boldsymbol{c}) \notin s \cap s_{next}$ (line 22), where $s_{next} = s \cup \mathsf{eff}'^{+}(op(\boldsymbol{c})) \setminus \mathsf{eff}'^{-}(op(\boldsymbol{c}))$ is the state resulting from the execution of $op(\boldsymbol{c})$ in $\mathcal{M}'(\mathcal{C})$. Let distinguish whether or not $p(\boldsymbol{c}) \in s_{next}$. If $p(\boldsymbol{c}) \notin s_{next}$ then $p(\boldsymbol{x}) \notin \mathsf{eff}'^{+}(op)$, as required. If $p(\boldsymbol{c}) \in s_{next}$ then $p(\boldsymbol{c}) \notin s$, which implies that that $p(\boldsymbol{x})$ is added to $\mathsf{eff}^{+}_{!}(op(\boldsymbol{c}))$ at line 20 and therefore $p(\boldsymbol{x}) \in \mathsf{eff}^{+}_{?!}(op)$, which contradicts the hypothesis that $p(\boldsymbol{x}) \notin \mathsf{eff}^{+}_{?!}(op)$. □

In the rest of the section, we study the properties of correctness and integrity for the learned models $\mathcal{M}$ and $\mathcal{M}^{-}_{?}$.

Theorem 2 (Correctness of $\mathcal{M}^{-}_{?}$)**.** *$\mathcal{M}^{-}_{?}$ correctly approximates $\mathcal{M}'$.*

Proof. Let s be a state of $\mathcal{M}^{-}_{?}(\mathcal{C}')$ for any set of constants $\mathcal{C}'$ possibly different from $\mathcal{C}$, and let $(s, op(\boldsymbol{c}), s^{-}_{?}) \in \delta^{-}_{?}$. By Lemmas 1-2, $\mathsf{pre}(op(\boldsymbol{c})) \supseteq \mathsf{pre}'(op(\boldsymbol{c}))$ and therefore there exists a tuple $(s, op(\boldsymbol{c}), s') \in \delta'$. We have that $s^{-}_{?} = s \cup \mathsf{eff}^{+}_{!}(op(\boldsymbol{c})) \setminus \mathsf{eff}^{-}_{!}(op) \setminus \mathsf{eff}^{-}_{?}(op)$. By Lemmas 1-4, $\mathsf{eff}^{+}_{!}(op(\boldsymbol{c})) \subseteq \mathsf{eff}'^{+}(op(\boldsymbol{c}))$; by Lemmas 1-5, $\mathsf{eff}'^{-}(op(\boldsymbol{c})) \subseteq \mathsf{eff}^{-}_{!?}(op(\boldsymbol{c})) = \mathsf{eff}^{-}_{!}(op(\boldsymbol{c})) \cup \mathsf{eff}^{-}_{?}(op(\boldsymbol{c}))$. Since $s' = s \cup \mathsf{eff}'^{+}(op(\boldsymbol{c})) \setminus \mathsf{eff}'^{-}(op(\boldsymbol{c}))$, it must be that $s^{-}_{?} \subseteq s'$. □

Theorem 3 (Correctness of $\mathcal{M}$)**.** *$\mathcal{M}(\mathcal{C})$ correctly approximates $\mathcal{M}'(\mathcal{C})$ from the final state of OLAM.*

Proof. Let s reachable from s_f in $\mathcal{M}(\mathcal{C})$. Suppose that $(s, op(\boldsymbol{c}), s'') \in \delta$. By Lemmas 1-2, $\mathsf{pre}(op(\boldsymbol{c})) \supseteq \mathsf{pre}'(op(\boldsymbol{c}))$ and therefore there exists a tuple $(s, op(\boldsymbol{c}), s') \in \delta'$. Suppose that $s'' \not\subseteq s'$, which implies that there is a $p(\boldsymbol{c}) \in s''$ which is not in s'. This can be caused by some missing negative effect in $\mathsf{eff}^{-}_{!}(op)$ or some extra positive effect in $\mathsf{eff}^{+}_{!}(op)$. The latter case is excluded by Lemma 4. Suppose that $p(\boldsymbol{x}) \in \mathsf{eff}'^{-}(op)$ but $p(\boldsymbol{x}) \notin \mathsf{eff}^{-}_{!}(op)$. From Lemma 5 we have that $\mathsf{eff}'^{-}(op) \subseteq \mathsf{eff}^{-}_{!?}(op)$. The fact that $\mathsf{eff}^{-}_{!}(op) \subseteq \mathsf{eff}'^{-}(op)$ implies $p(\boldsymbol{x}) \in \mathsf{eff}^{-}_{?}(op)$. Since s is reachable from the final state s_f via $\mathcal{M}$, then condition (4) must be false, which means that $\mathsf{eff}^{-}_{?}(op(\boldsymbol{c})) \cap s = \emptyset$, and therefore that $p(\boldsymbol{c}) \notin s$. Therefore if $p(\boldsymbol{c}) \in s''$, then it has been added by $p(\boldsymbol{x}) \in \mathsf{eff}^{+}_{!}(op) \subseteq \mathsf{eff}'^{+}(op)$. But this contradicts the fact that $p(\boldsymbol{x}) \in \mathsf{eff}'^{-}(op)$. □

Lemma 6. *At any execution step of OLAM if $\phi \in \mathsf{pre}_{\perp}(op)$ and $\phi \neq \emptyset$ then $\phi \cap \mathsf{pre}'(op) \neq \emptyset$.*

Proof. If $\phi \in \mathsf{pre}_{\perp}(op)$ and $\phi \neq \emptyset$, then ϕ has been added because of the failure of an action $op(\boldsymbol{c})$ in a state s and $\phi = \{p(\boldsymbol{x}) \in \mathsf{pre}(op) \mid p(\boldsymbol{c}) \notin s\}$. The failure of $op(\boldsymbol{c})$ in s implies that there is a $p(\boldsymbol{x}) \in \mathsf{pre}'(op)$ such that $p(\boldsymbol{c}) \notin s$. The fact that $\mathsf{pre}(op) \supseteq \mathsf{pre}'(op)$ implies that $p(\boldsymbol{x}) \in \phi$ and therefore $p(\boldsymbol{x}) \in \phi \cap \mathsf{pre}'(op)$. Hence we can conclude that $\phi \cap \mathsf{pre}'(op) \neq \emptyset$. □

Theorem 4 (Integrity of $\mathcal{M}$)**.** *$\mathcal{M}(\mathcal{C})$ integrally approximates $\mathcal{M}'(\mathcal{C})$ from the final state of OLAM.*

Proof. Suppose that s is reachable form the final state of OLAM via $\mathcal{M}(\mathcal{C})$ and that $op(\boldsymbol{c})$ is executable from s according to $\mathcal{M}'(\mathcal{C})$, i.e., that $\mathsf{pre}'(op(\boldsymbol{c})) \subseteq s$. First, let us show that $op(\boldsymbol{c})$ is also executable by $\mathcal{M}(\mathcal{C})$, i.e., that $\mathsf{pre}(op(\boldsymbol{c})) \subseteq s$. Suppose the contrary, i.e., that $\mathsf{pre}(op(\boldsymbol{c})) \setminus s \neq \emptyset$. Since s is reachable from s_f with $\mathcal{M}(\mathcal{C})$, the fact that OLAM terminates at s_f implies that condition (2) is false. This implies that $\mathsf{pre}(op(\boldsymbol{c})) \setminus s \in \mathsf{pre}_{\perp}(op(\boldsymbol{c}))$, which implies that $\phi = \{p(\boldsymbol{x}) \in \mathsf{pre}(op) \mid p(\boldsymbol{c}) \notin s\} \in \mathsf{pre}_{\perp}(op)$. Furthermore ϕ is not empty since $\mathsf{pre}(op(\boldsymbol{c})) \setminus s \neq \emptyset$. By Lemma 6 we have that there is $p(\boldsymbol{x}) \in \mathsf{pre}'(op)$ such that $p(\boldsymbol{c}) \notin s$, which implies that $op(\boldsymbol{c})$ is not executable in s by $\mathcal{M}'(\mathcal{C})$, which is a contradiction.

Let $(s, op(\boldsymbol{c}), s'') \in \delta$ and $(s, op(\boldsymbol{c}), s') \in \delta'$. Suppose by contradiction that $s' \not\subseteq s''$, which implies that there is a $p(\boldsymbol{c}) \in s'$ which is not in s''. This can be caused by some missing positive effect in $\mathsf{eff}^{+}_{!}(op)$ or some extra negative effect in $\mathsf{eff}^{-}_{!}(op)$. The latter case is excluded by Lemma 4. Suppose that $p(\boldsymbol{x}) \in \mathsf{eff}'^{+}(op)$ but $p(\boldsymbol{x}) \notin \mathsf{eff}^{+}_{!}(op)$. From Lemma 5 we have that $\mathsf{eff}'^{+}(op) \subseteq \mathsf{eff}^{+}_{!?}(op)$. The fact that $\mathsf{eff}^{+}_{!}(op) \subseteq \mathsf{eff}'^{+}(op)$ implies $p(\boldsymbol{x}) \in \mathsf{eff}^{+}_{?}(op)$. Since s is reachable from the final state via $\mathcal{M}$, condition (3) must be false and therefore $\mathsf{eff}^{+}_{?}(op(\boldsymbol{c})) \subseteq s$. This implies that $p(\boldsymbol{c}) \in s$. Therefore if $p(\boldsymbol{c}) \notin s''$, then it has been deleted by $p(\boldsymbol{x}) \in \mathsf{eff}^{-}_{!}(op) \subseteq \mathsf{eff}'^{-}(op)$. But this contradicts the fact that $p(\boldsymbol{x}) \in \mathsf{eff}'^{+}(op)$. □

The learned model $\mathcal{M}$ approximates the GTM from the final state s_f of OLAM *both* correctly and integrally. This implies that all and only the valid plans computed from s_f via $\mathcal{M}$ are valid plans from s_f via the GTM. Therefore, if a complete algorithm fails to reach a given set of goals from s_f via $\mathcal{M}$, then the goals cannot be reached also via the GTM.

Experiments

We evaluate the effectiveness of OLAM for online learning planning domains on 23 planning domains, including the domains from the learning tracks of the past IPCs and the domains used by Aineto, Jiménez Celorrio, and Onaindia (2019). For each domain, using an available problem generator, we randomly generated 10 small or middle-size instances with a number of objects ranging from 3 to 241 and consequently a number of potential grounded actions ranging from 12 to about $3.16 \cdot 10^{6}$. For every domain OLAM is

Table 1: Number of instances used to learn $\mathcal{M}$ (column 2), precision and recall over the preconditions, positive and negative effects of $\mathcal{M}$ (columns 3–8), overall precision and recall of $\mathcal{M}$ (columns 9-10).

Domain	$\#I$	P_{pre}	R_{pre}	P_{eff^+}	R_{eff^+}	P_{eff^-}	R_{eff^-}	P	R
barman	4	0.95	1	1	1	1	1	0.97	1
blocksworld	1	1	1	1	1	1	1	1	1
depots	1	0.94	1	1	1	1	1	0.97	1
driverlog	2	0.88	1	1	1	1	1	0.93	1
elevators	3	0.81	1	1	1	1	1	0.88	1
ferry	1	0.88	1	1	1	1	1	0.94	1
floortile	1	0.71	1	1	1	1	1	0.83	1
gold-miner	2	0.68	1	1	1	1	1	0.80	1
grid	2	0.71	1	1	1	1	1	0.82	1
gripper	1	1	1	1	1	1	1	1	1
hanoi	1	0.80	1	1	1	1	1	0.88	1
matching-bw	3	0.97	1	1	1	1	1	0.99	1
miconic	1	1	1	1	1	1	1	1	1
n-puzzle	1	0.75	1	1	1	1	1	0.88	1
nomystery	1	0.75	1	1	1	1	1	0.85	1
parking	2	0.78	1	1	1	1	1	0.89	1
rover	5	0.78	1	1	0.65	1	0.54	0.83	0.84
satellite	1	1	1	1	1	1	1	1	1
sokoban	1	0.80	1	1	1	1	1	0.89	1
spanner	1	0.90	1	1	1	1	1	0.94	1
tpp	3	0.94	1	1	1	1	1	0.97	1
transport	1	0.91	1	1	1	1	1	0.95	1
zenotravel	1	1	1	1	1	1	1	1	1

run on all the generated problem instances, from the smallest to the largest. On the first instance, OLAM takes in input the empty set of preconditions, positive and negative effects; for the successive runs, OLAM takes in input the planning domain $\mathcal{M}$ learned at the previous run. In OLAM, the calls EXECUTE and OBSERVE (lines 17-18) are implemented by a simulator of the IPC domain. Notice that the transition function of such a model is not known by the agent, who can only ask to execute actions and observe the current state. For function PLAN (line 8), we adopt FASTDOWNWARD (Helmert 2006) using lazy greedy best-first search with the FF heuristic (Hoffmann 2001) and the context-enhanced additive heuristic (Helmert and Geffner 2008), and with a 60 seconds timeout. FastDownward supports goal formulas expressed as a disjunction of conjunctions of atoms. As input goal formula we give the disjunction in Equation (1) (without the subsumed disjuncts). All experiments were run on an Intel Xeon Skylake 2.3 GHz with 8 cores and 64 GB of RAM.

The learned planning domain is compared with the GTM, as done by Aineto, Jiménez Celorrio, and Onaindia (2019), by precision and recall measures. Given a learned model $\mathcal{M}$ and GTM $\mathcal{M}'$, we define precision and recall for preconditions (P_{pre}, R_{pre}), positive and negative effects (P_{eff^-}, P_{eff^+}, R_{eff^-}, R_{eff^+}). Specifically, P_{pre} and R_{pre} are defined as follows:

$$P_{\mathsf{pre}} = \frac{\sum_{op}|\mathsf{pre}(op) \cap \mathsf{pre}'(op)|}{\sum_{op}|\mathsf{pre}(op)|} \quad R_{\mathsf{pre}} = \frac{\sum_{op}|\mathsf{pre}(op) \cap \mathsf{pre}'(op)|}{\sum_{op}|\mathsf{pre}'(op)|}.$$

Intuitively, they measure the (relative) amount of *extra* learned preconditions w.r.t. the GTM, and the (relative) amount of *missing* preconditions w.r.t. the GTM, respectively. The lower these amounts, the greater the measures. Similarly we define precision and recall for eff^- and eff^+. If the precision and recall measures for pre, eff^- and eff^+ is 1, then the learned model is exactly the same as in the GTM for pre, eff^- and eff^+, respectively. The *overall* precision P and recall R are defined considering pre, eff^-, eff^+ together. I.e.,

$$P = \frac{\sum_{op}|\mathsf{pre}(op) \cap \mathsf{pre}(op)'| + |\mathsf{eff}^+(op) \cap \mathsf{eff}'^+(op)'| + |\mathsf{eff}^-(op) \cap \mathsf{eff}'^-(op)'|}{\sum_{op}|\mathsf{pre}(op)| + |\mathsf{eff}^+(op)| + |\mathsf{eff}^-(op)|},$$

and similarly for R.

Table 1 summarizes the efficacy of $\mathcal{M}$ w.r.t. the GTM in terms of precision and recall. By construction of sets $\mathsf{pre}(op)$ and $\mathsf{eff}^{+/-}_{!}(op)$ of every operator op, R_{pre}, P_{eff^+}, and P_{eff^-} of $\mathcal{M}$ must be equal to 1, i.e., there is no missing precondition and extra effect in the learned model $\mathcal{M}$ w.r.t. the GTM. This is confirmed by the results in Table 1. Moreover, P_{pre} is always quite high, although usually lower than 1, i.e., there are few extra preconditions in the learned model w.r.t. the GTM. The extra learned preconditions are static predicates such that, when the action is grounded, the corresponding grounded preconditions are true in all the states reachable from the initial state. This prevents the remotion of these extra preconditions from a correct learned model, like $\mathcal{M}$. The recall over the positive/negative effects is always equal to 1 for every domain but ROVER, i.e., there are no missing effects (except for ROVER) in the learned model w.r.t. the GTM. Finally, the overall precision and recall of $\mathcal{M}$ are quite close or equal to 1.

The results in Table 1 also show that domain $\mathcal{M}$ can be learned using very few problems, often using only a single problem. Note that such a domain is learned by few small problems, and it does not mention their constants, i.e., it is general and hence suitable even for much larger problems. This shows that overall OLAM is able to effectively generalize between the experience derived from small environments and the future experience in large environments.

We also study the efficacy of $\mathcal{M}^-_?$ w.r.t. the GTM. The difference between the learned models $\mathcal{M}$ and $\mathcal{M}^-_?$ consists in the fact that $\mathcal{M}^-_?$ also includes set $\mathsf{eff}^-_?(op)$ as negative effects of an operator op. Therefore, the precision and the recall over the preconditions and the positive effects of $\mathcal{M}^-_?$ are the same as in Table 1. Table 2 gives the precision and recall over the negative effects of $\mathcal{M}^-_?$ and over all domain $\mathcal{M}^-_?$. For this study we consider $\mathcal{M}^-_?$ with and without assuming $\mathsf{eff}'^-(op) \subseteq \mathsf{pre}'(op)$, i.e., when this assumption is made, the atoms in $\mathsf{eff}^-_?(op)$ that are not in the preconditions of an operator are removed. By construction of set $\mathsf{eff}^-_{!?}$, R_{eff^-} must be equal to 1. Surprisingly, this is false for domains GOLD-MINER and ROVER. The reason why this happens is that for these domains an assumption of ours does not hold: ROVER is a special domain including operators with inconsistent effects, i.e., $\mathsf{eff}'^+(op) \cap \mathsf{eff}'^-(op) \neq \emptyset$, for some operators. For GOLD-MINER, the assumption $\mathsf{eff}'^-(op) \subseteq \mathsf{pre}'(op)$ does not hold. This assumption is violated also in domains PARKING, SATELLITE and MATCHING-BW, but

Table 2: Precision and recall over the negative effects of $\mathcal{M}^{-}_{?}$ and overall model $\mathcal{M}^{-}_{?}$ with the assumption $\mathsf{eff}'^{-}(op) \subseteq \mathsf{pre}'(op)$ (columns 2–5), and without this assumption (columns 6–9).

	wth assumption				without assumption			
Domain	P_{eff^-}	R_{eff^-}	P	R	P_{eff^-}	R_{eff^-}	P	R
barman	1	1	0.97	1	0.24	1	0.56	1
blocksworld	1	1	1	1	0.43	1	0.69	1
depots	1	1	0.97	1	0.56	1	0.80	1
driverlog	1	1	0.93	1	0.23	1	0.53	1
elevators	1	1	0.88	1	0.15	1	0.42	1
ferry	1	1	0.94	1	0.50	1	0.75	1
floortile	1	1	0.83	1	0.10	1	0.29	1
gold-miner	1	0.82	0.80	0.95	0.18	1	0.41	1
grid	1	1	0.82	1	0.28	1	0.55	1
gripper	1	1	1	1	1	1	1	1
hanoi	1	1	0.88	1	1	1	0.88	1
matching-bw	1	1	0.99	1	0.32	1	0.65	1
miconic	1	1	1	1	0.23	1	0.62	1
n-puzzle	1	1	0.88	1	0.50	1	0.70	1
nomystery	1	1	0.85	1	0.10	1	0.30	1
parking	1	1	0.89	1	0.35	1	0.60	1
rover	1	0.54	0.83	0.84	0.16	0.54	0.55	0.84
satellite	1	1	1	1	0.67	1	0.92	1
sokoban	1	1	0.89	1	0.25	1	0.53	1
spanner	1	1	0.94	1	0.40	1	0.70	1
tpp	1	1	0.97	1	0.15	1	0.42	1
transport	1	1	0.95	1	0.33	1	0.65	1
zenotravel	1	1	1	1	0.33	1	0.67	1

for these domains there is no missing negative effect in $\mathcal{M}^{-}_{!}$, since OLAM on line 21 learns $\mathsf{eff}^{-}_{!}$ regardless of this assumption. Interestingly, P^{-}_{eff} with this assumption is always equal to 1, while without the assumption it is almost always quite low. This gap gives evidence that such an assumption can be very useful for removing extra negative effects from the learned domain.

We compare OLAM with a version of the algorithm that explores the world randomly. The random strategy reaches an average precision and recall of 0.45 and 0.63, against the average precision and recall of 0.99 and 0.92 obtained by OLAM, which shows that the usage of the learned model is extremely helpful.

In the last experiment we compare the online learning of OLAM with the offline learning method proposed by Fama (Aineto, Jiménez Celorrio, and Onaindia 2019). Although the output of FAMA is the same as OLAM, i.e., the learned planning domain, it is worth noting that the input is quite different. In addition to our input knowledge, Fama takes as input a set of plans with their state trajectories. Since OLAM does not support partial observability, we set Fama for working in a fully observable environment, and considered the same sets of plan traces and planning domains (but VISITALL and ZENOTRAVEL) as in (Aineto, Jiménez Celorrio, and Onaindia 2019). The set of plan traces consists of 10 traces with 10 states; the set of planning domains does not contain ZENOTRAVEL and VISITALL, because the distributed version of Fama finds no solution for ZENOTRAVEL, and there is no problem generator available for VISITALL. Since Fama adopts the assumption $\mathsf{eff}'^{-}(op) \subseteq \mathsf{pre}'(op)$ for any operator op, we compared the planning domain derived from Fama with $\mathcal{M}^{-}_{?}$ using the same assumption. We obtained similar results from the comparison between Fama and the other learned domain $\mathcal{M}$.

Table 3: CPU-seconds, precision and recall of OLAM (columns 2–4) and Fama (columns 5–7); difference between number of actions executed by Fama and OLAM (column 8): negative values mean that OLAM executes fewer actions. Bold values indicate best results.

	OLAM			Fama			
Domain	Time	P	R	Time	P	R	Δ acts
blocksworld	5.03	**1**	**1**	510	**1**	**1**	-80
driverlog	20.42	**0.93**	**1**	349	0.79	0.85	-43
ferry	7.54	**0.94**	**1**	267	0.80	0.93	-85
floortile	47.34	**0.83**	**1**	517	0.82	0.78	-15
grid	36.92	**0.82**	**1**	306	0.81	0.74	-1
gripper	3.50	**1**	**1**	165	0.86	0.93	-89
hanoi	2.38	**0.88**	**1**	818	**0.88**	0.86	-96
miconic	4.24	**1**	**1**	200	0.81	**1**	-78
n-puzzle	1.97	**0.88**	**1**	23	0.86	**1**	-91
parking	183.94	**0.89**	**1**	895	0.84	0.84	-47
rover	154.10	**0.83**	**0.84**	629	0.51	0.53	175
satellite	11.26	**1**	**1**	65	0.70	0.89	-54
transport	74.98	**0.95**	**1**	280	0.80	0.89	-32

Table 3 compares OLAM and Fama. OLAM obtains better or equal precision and recall, and generally it is also much faster. In all the domains but ROVER, OLAM executes less actions than Fama. We think that the difference for ROVER is related to the consistent-effects assumption made in OLAM that in ROVER does not hold. Overall, learning the planning domain online is much more effective than learning it offline. In our online approach, indeed, the agent selects the goals to reach and actions to execute to optimize learning, while in offline approaches actions are provided in the input traces.

Conclusions

The paper proposes an online algorithm, OLAM, for learning PDDL planning domain under the assumption of full observability. OLAM incrementally learns an action model by selecting goals to reach and actions to execute that allow to acquire useful information about the operators. The paper shows some important theoretical properties of OLAM concerning the completeness and integrity of the learned models. An implementation of OLAM shows good learning performance on a large set of benchmarks from the IPCs, and outperforms a state-of-the-art method for learning action models offline. OLAM works with full observability; extension to partial observability is part of the future work.

Acknowledgements

This work is partially supported by the EU ICT-48 2020 project TAILOR (No. 952215).

References

Aineto, D.; Jiménez, S.; and Onaindia, E. 2018. Learning STRIPS action models with classical planning. In *ICAPS*.

Aineto, D.; Jiménez Celorrio, S.; and Onaindia, E. 2019. Learning action models with minimal observability. *Artif. Intell.* 275: 104 – 137. ISSN 0004-3702.

Amir, E.; and Chang, A. 2008. Learning Partially Observable Deterministic Action Models. *J. Artif. Intell. Res.* 33: 349–402.

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

Certicky, M. 2014. Real-time action model learning with online algorithm 3SG. *Applied Artificial Intelligence* 28(7): 690–711.

Cresswell, S.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using *LOCM*. *Knowledge Eng. Review* 28(2): 195–213.

Fern, A.; Yoon, S. W.; and Givan, R. 2004. Learning Domain-Specific Control Knowledge from Random Walks. In *ICAPS*.

García-Martínez, R.; and Borrajo, D. 2000. An Integrated Approach of Learning, Planning, and Execution. *J. Intell. Robotic Syst.* 29(1): 47–78.

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

Gil, Y. 1994b. Learning New Planning Operators by exploration and Experimentation. In *AAAI*.

Helmert, M. 2006. The Fast Downward Planning System. *J. Artif. Intell. Res.* 26: 191–246.

Helmert, M.; and Geffner, H. 2008. Unifying the Causal Graph and Additive Heuristics. In *ICAPS*, 140–147.

Hoffmann, J. 2001. FF: The fast-forward planning system. *AI magazine* 22(3): 57–57.

Kitano, H.; and Tadokoro, S. 2001. RoboCup Rescue: A Grand Challenge for Multiagent and Intelligent Systems. *AI Mag.* 22(1): 39–52.

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

Mourão, K.; Zettlemoyer, L. S.; Petrick, R. P. A.; and Steedman, M. 2012. Learning STRIPS Operators from Noisy and Incomplete Observations. In *UAI*.

Newton, M. A. H.; Levine, J.; Fox, M.; and Long, D. 2007. Learning Macro-Actions for Arbitrary Planners and Domains. In *ICAPS*.

Rodrigues, C.; Gérard, P.; and Rouveirol, C. 2010. Incremental Learning of Relational Action Models in Noisy Environments. In *ILP*.

Rodrigues, C.; Gérard, P.; Rouveirol, C.; and Soldano, H. 2010. Incremental Learning of Relational Action Rules. In *ICMLA*.

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

Stachniss, C.; Leonard, J. J.; and Thrun, S. 2016. Simultaneous Localization and Mapping. In *Springer Handbook of Robotics*, Springer Handbooks, 1153–1176. Springer.

Sutton, R. S.; and Barto, A. G. 1998. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press.

Walsh, T. J.; and Littman, M. L. 2008. Efficient learning of action schemas and web-service descriptions. In *AAAI*.

Wang, X. 1996. Planning While Learning Operators. In *AAAI*.

Xu, J. Z.; and Laird, J. E. 2010. Instance-Based Online Learning of Deterministic Relational Action Models. In *AAAI*.

Xu, J. Z.; and Laird, J. E. 2011. Combining Learned Discrete and Continuous Action Models. In *AAAI*.

Xu, J. Z. Y.; and Laird, J. E. 2013. Learning Integrated Symbolic and Continuous Action Models for Continuous Domains. In *AAAI*.

Yang, Q.; Wu, K.; and Jang, Y. 2007. Learning action models from plan examples using weighted MAX-SAT. *Artif. Intell.* 171: 107–143.

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

Zhuo, H. H.; Yang, Q.; Hu, D. H.; and Li, L. 2010. Learning complex action models with quantifiers and logical implications. *Artif. Intell.* 174(18): 1540–1569.