# Offline Learning of Planning Domains with Subsymbolic Predicates Invention

**Leonardo Lamanna**

Augmented Intelligence Center, Fondazione Bruno Kessler, Trento, Italy
llamanna@fbk.eu

**Abstract**

The manual specification of planning domains is a notoriously time-consuming and error-prone process that requires extensive knowledge of the environment. Several approaches have been proposed for learning planning domains from execution traces, i.e., sequences of state observations obtained by executing actions, dealing with different assumptions about the observability of states and actions. However, most of the approaches either assume the planning domains to be specified using a predefined set of predicates, or do not consider traces with subsymbolic state observations (e.g., RGB images). In this paper, we propose an approach for learning planning domains from fully observable traces where state observations are grid-based RGB images. The proposed approach invents symbolic predicates that are grounded to the trace subsymbolic observations, and learns a symbolic transition model defined on the state space induced by the invented predicates. We experiment with the proposed approach in planning domains provided in previous International Planning Competitions, adopting an RGB grid-based representation of states. We experimentally show the effectiveness of the proposed approach by planning with the learned domains to achieve goals in previously unseen environments.

## 1 Introduction

Automated planning techniques are powerful tools for decision-making agents that needs to accomplish long-horizon tasks in complex environments (Ghallab, Nau, and Traverso 2004). These techniques assume a model of the dynamics of an agent environment is available a priori. Such models are typically handcrafted by humans, which requires domain experts. Moreover, manually specifying planning domain models is an error-prone and time-consuming process. Indeed, when specifying a model of an environment dynamics, there is the necessity of describing how the environment state evolves when an agent executes actions, abstracting away details of the environment state that are irrelevant for the achievement of an agent's goals. To overcome the limitations of manually specifying planning domains, several approaches have been proposed for learning planning domains from a given set of execution traces, which are subsequent observations of environment states obtained while executing a sequence of actions. The proposed approaches focus on different assumptions about the observability of the environment states and executed actions. For example, some approaches learn the set of preconditions and effects of planning operators in fully observable (Mordoch et al. 2024), partially observable (Aineto, Celorrio, and Onaindia 2019), or noisy environments (Lamanna and Serafini 2024). Most of the proposed approaches assume the observed environment states to be represented by means of a known set of domain predicates. Interestingly, recent approaches do not assume the domain predicates to be known beforehand (Asai and Muise 2020; Gösgens, Jansen, and Geffner 2025), and some approaches (Asai et al. 2022) learn planning domains from state observations consisting of RGB images, possibly without any knowledge about the executed actions. While the problem of learning planning domains has been widely recognized to be a compelling challenge, see, e.g. (Callanan et al. 2022) for a literature review, current approaches still suffer from several drawbacks. Firstly, they mostly focus on learning from symbolic observations of environment states, whereas, in real-world environments, agents typically perceive their environments through low-level sensors that provide subsymbolic observations of the environment states. Interestingly, some approaches learn planning domains from subsymbolic observations (see, e.g., (Xi, Gould, and Thiébaux 2024)). However, current approaches for learning planning domains from subsymbolic observations either assume the predicates set to be given or do not learn a planning domain that can be used for planning in unseen environments. Moreover, no approach is able to invent symbolic predicates while learning a mapping from predicates and subsymbolic observations.

In this paper, we propose an approach, namely Subsymbolic Learning of Action Models (S-LAM), that aims to overcome the abovementioned limitations. We focus on offline learning planning domains from execution traces obtained by observing an agent acting in the environment, where state observations are RGB images. We assume the observations are object-centric, i.e., they represent a set of objects with associated bounding boxes. This assumption is reasonable as there exist pretrained object detectors (e.g., (Kirillov et al. 2023)) that recognize objects and their bounding boxes in perceptions of real-world, open-ended environments. The main contributions of this work are as follows:

1. We propose an approach named S-LAM for offline learning planning domains from subsymbolic, object-centric observations, without assuming the domain predicates to

be known a priori, and learning symbolic predicates that are grounded to subsymbolic observations.

2. We provide an algorithm implementation for S-LAM, where the learned planning domains are specified as relational Markov Decision Processes (MDPs), and planning is performed using a Monte Carlo tree search based algorithm (Silver and Veness 2010).
3. We evaluate S-LAM on four variants of previous IPC planning domains with grid-based state images, and experimentally evaluate the planning effectiveness with the domains learned by S-LAM, showing they are effective to solve planning problems in unseen environments, and highlighting their limitations in complex problems.

We discuss related approaches for offline learning planning domains in Section 2. Preliminary notions and assumptions are stated in Section 3, whereas a detailed explanation of S-LAM is reported in Section 4. In Section 5, we describe the considered grid-based variants of $4$ IPC domains, and evaluate the planning performance with the domains learned by S-LAM. Finally, Section 6 discusses limitations and outlines directions for future research.

## 2 Related Work

Most of the approaches for learning planning domains have been focused on *offline* learning, assuming an input set of execution traces, where each trace is a sequence of states obtained by executing subsequent actions.

A large amount of work considered learning planning domains in fully observable environments (Stern and Juba 2017; Mordoch, Juba, and Stern 2023; Mordoch et al. 2024; Aineto and Scala 2024), or partially observable environments (Wu, Yang, and Jiang 2007; Aineto, Celorrio, and Onaindia 2019; Le, Juba, and Stern 2024; Lamanna et al. 2025).

Few approaches learn from traces with noisy states, i.e., where the observed truth-value of a literal in a state may differ from its actual truth-value in such a state. ALICE (Mourão et al. 2012) deal with both noisy and partially observable states, ALICE trains a set of classifiers for predicting the action effects, and then extracts an action model from the classifiers in the form of a set of logical rules. Similar assumptions hold for PlanMiner (Segura-Muros, Pérez, and Fernández-Olivares 2018), which applies a set of inductive logic rules based on a dataset of transitions for every operator. (Lamanna and Serafini 2024) considers noisy and fully observable states, and employs a probabilistic approach that, for every domain operator and atom, estimates the probability the atom being a precondition effect conditioned by the observations in the input traces.

All the abovementioned work assumes that the observed states are represented by means of a set of domain symbolic predicates known beforehand. In contrast, S-LAM does not make such an assumption. A recent work (Gösgens, Jansen, and Geffner 2025) learns action models from traces with fully unobserved states and fully observable actions, without assuming the set of domain predicates to be known a priori. With respect to (Gösgens, Jansen, and Geffner 2025), S-LAM considers execution traces with state subsymbolic observations, and invents predicates that are grounded to such subsymbolic observations, learning to map the observations into the symbolic space induced by the learned predicates, and viceversa.

The work by (Asai 2019; Asai et al. 2022) proposed LatPlan, an approach for learning domain models from subsymbolic perceptions (i.e., RGB images). LatPlan proposed a neuro-symbolic neural network architecture, which, based on Variational Auto Encoders (Pu et al. 2016), learns a latent binary representation of the state corresponding to a given image. LatPlan learns both an encoder that maps images into symbolic states and a decoder that maps symbolic states into images. The learned encoder and decoder allow LatPlan to ground the invented symbols into subsymbolic perceptions, as in S-LAM. However, LatPlan learns symbolic state representations that are propositional, and the encoder and decoder networks are not shown to generalize to environments different from the ones from which (possibly a large amount of) training data is obtained. As in LatPlan, S-LAM learns both a mapping from subsymbolic observations to the invented predicates, and viceversa. Differently from LatPlan, S-LAM invents predicates that are lifted – a predicate is grounded to observations of possibly different objects. Consequently, the learned domains can be used for planning in unseen environments where objects provide observations similar to the ones in the execution traces. Moreover, S-LAM does not require a large amount of data to learn from, as it can, in principle, learn from a few observations of state images before and after executing an action. (Xi, Gould, and Thiébaux 2024) proposed ROSAME, a neuro-symbolic approach for learning planning domains from RGB images. ROSAME overcomes some limitations of LatPlan, such as, e.g., learning to encode (resp. decode) RGB images into (resp. from) symbolic predicates that are lifted. However, ROSAME requires the set of symbolic predicates to be known beforehand, and the first and last RGB images of each execution trace to be labeled with the truth-values of such predicates; S-LAM does not require the set of predicates to be given, and is a fully unsupervised approach for which no observation needs to be labeled.

The work by (Konidaris, Kaelbling, and Lozano-Pérez 2018) learns probabilistic planning domains (expressed in PPDDL, (Younes and Littman 2004)) from execution traces with subsymbolic observations. Interestingly, as S-LAM, the approach in (Konidaris, Kaelbling, and Lozano-Pérez 2018) invents symbolic predicates based on the subsymbolic observations. However, the considered observations are based on simple hand-crafted state features (e.g., $x$ and $y$ agent position), and the learned domains are grounded, preventing from planning in environments different from the ones from which the execution traces are produced. On the opposite, S-LAM learns from complex subsymbolic observations such as RGB images, and the invented symbolic predicates are lifted, which allows the learned domain to be reused in different environments.

## 3 Preliminaries

Let $\mathcal{C}$ be a set of constants identifying objects, and $\mathcal{P}$ a set of predicates, where every $p_i \in P$ is associated with arity

$\alpha \in \mathbb{N}$; a predicate with arity $\alpha$ is denoted by $p^{(\alpha)}$, and the set of predicates with arity $\alpha$ by $\mathcal{P}^{(\alpha)} \subseteq \mathcal{P}$. Unary predicates ($\alpha = 1$) describe object properties, and predicates with arity $\alpha > 1$ describe object relations. For example, in the blocksworld domain, the unary predicate ONTABLE applied to an object $block_0$, i.e., ONTABLE($block_0$), indicates that the $block_0$ is on the table. We denote by $\mathcal{P}(\mathcal{C})$ the set of ground atoms obtained by instantiating all predicates in $\mathcal{P}$ with constants in $\mathcal{C}$. For example, when $\mathcal{C} = \{block_0, block_1\}$ and $\mathcal{P} = \{\text{ONTABLE}\}$, then $\mathcal{P}(\mathcal{C}) = \{\text{ONTABLE}(block_0), \text{ONTABLE}(block_1)\}$.

A *discrete and deterministic* planning domain $\mathcal{D}$ is a tuple $\langle \mathcal{A}, \mathcal{S}, \delta \rangle$ where $\mathcal{A}$ is a set of discrete actions, $\mathcal{S}$ is a set of discrete states, and $\delta : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is a deterministic transition function such that $\delta(s, a)$ is the state $s'$ reached after executing action $a$ in state $s$. An action $a \in \mathcal{A}$ is defined by an action schema $\langle \mathsf{param}_a, \mathsf{pre}_a, \mathsf{eff}_a \rangle$ where $\mathsf{param}_a$ is the sequence of action parameters, $\mathsf{pre}_a$ the set of preconditions, and $\mathsf{eff}_a$ the set of effects; additionally, $\mathsf{eff}_a = \mathsf{eff}_a^+ \cup \mathsf{eff}_a^-$, with the set $\mathsf{eff}_a^+$ of *positive* effects and $\mathsf{eff}_a^-$ of *negative* effects. The preconditions must hold in a state for the action to be executable, and the effects describes how the state evolves after executing the action. For example, in the blocksworld domain, the action PUTDOWN($block_0$) can be executed in a state $s \in \mathcal{S}$ where the agent holds $block_0$, i.e., $\mathsf{pre}_a = \{\text{HOLDING}(block_0)\}$; the negative effect is that the agent no longer holds the block, i.e., $\mathsf{eff}_a^- = \{\text{HOLDING}(block_0))\}$; the positive effects are that the block is on the table (ONTABLE($block_0$)), the block is clear (CLEAR($block_0$)), and the agent hand is empty (HANDEMPTY()).

An action $a \in \mathcal{A}$ is *grounded* when $\mathsf{param}_a$ is a sequence of constants in $\mathcal{C}$, and lifted when $\mathsf{param}_a$ is a sequence of variables that can be instantiated with a sequence of constants in $\mathcal{C}$; similarly for the preconditions and effects. Throughout the paper, for an action $a$ where $\mathsf{param}_a$ is a sequence of constants, we say that the constants are involved in the action. A grounded action $a \in \mathcal{A}$ is executable in a state $s \in \mathcal{S}$ if $s \models \mathsf{pre}_a$, and the state $s' \in \mathcal{S}$ reached after executing $a$ in $s$ is $s' = s \setminus \mathsf{eff}_a^- \cup \mathsf{eff}_a^+$. A plan $\pi$ is a sequence of ground actions, and an execution trace is a sequence $\langle s_0, a_1, s_1, a_2, \ldots, a_n, s_n \rangle$ obtained by executing a plan $\pi = \langle a_1, a_2, \ldots, a_n \rangle$ from the state $s_0$.

A planning problem is a tuple $\langle \mathcal{D}, s_0, \mathcal{G} \rangle$ where $\mathcal{D}$ is a planning domain, $s_0 \in \mathcal{S}$ is an initial state, and $\mathcal{G} \subseteq \mathcal{S}$ is a subset of goal states that can be specified by a first-order logic formula $g$ such that $s_g \models g$ for every $s_g \in \mathcal{G}$. A solution to a planning problem $\langle \mathcal{D}, s_0, \mathcal{G} \rangle$ is a plan $\pi = \langle a_0, \ldots, a_n \rangle$ such that executing $\pi$ from $s_0$ leads to $s_n \in \mathcal{G}$.

The above definitions model deterministic and discrete planning domains. A widely adopted formalism to model possibly stochastic planning domains are MDPs.

**Definition 1 (Markov Decision Process)** *A Markov Decision Process (MDP) $\mathcal{M}$ is a tuple $\langle \hat{\mathcal{S}}, \hat{\mathcal{A}}, \mathcal{T}, \mathcal{R}, \gamma \rangle$ composed by a set $\hat{\mathcal{S}}$ of* states*, a set $\hat{\mathcal{A}}$ of* actions*, a probabilistic* transition function $\mathcal{T} : \hat{\mathcal{S}} \times \hat{\mathcal{A}} \times \hat{\mathcal{S}} \to [0, 1]$*, and a* reward function $\mathcal{R} : \hat{\mathcal{S}} \times \hat{\mathcal{A}} \times \hat{\mathcal{S}} \to \mathbb{R}^+$.

For every $s, s' \in \hat{\mathcal{S}}$ and $a \in \hat{\mathcal{A}}$, the transition function $\mathcal{T}(s, a, s')$ expresses the probability $Pr(s'|s, a)$ of reaching $s'$ by executing $a$ in $s$, and $\sum_{s' \in \hat{\mathcal{S}}} \mathcal{T}(s, a, s') = 1$. The reward function $\mathcal{R}$ returns the reward $\mathcal{R}(s, a, s')$ obtained when reaching $s'$ after executing $a$ in $s$; $\gamma \in [0, 1]$ is commonly referred to as discount factor.

A *Relational MDP* (RMDP) (van Otterlo 2012) is an MDP defined over a relational state space $\hat{\mathcal{S}}$ induced by a set of predicates $\mathcal{P}$ and constants $\mathcal{C}$; the state space of an RMDP is the set $\hat{\mathcal{S}} = 2^{\mathcal{P}(\mathcal{C})}$ of truth assignments to the ground atoms in $\mathcal{P}(\mathcal{C})$.

## Problem and Assumptions

We consider the problem of offline learning planning domains from a set of execution traces obtained by observing an agent acting in a fully observable environment. We do not assume the states of the execution traces to be specified in terms of predicates and constants, but rather deal with the problem of learning from subsymbolic states.

In this work, we consider execution traces where states are RGB images, and we assume an agent is capable to recognize the bounding boxes of the state objects observed in the image. This assumption is reasonable in open-ended and fully observable environments, as an agent perceives its environment through sensors (e.g., an RGB camera) and there exist open-ended, pretrained object detectors capable of providing bounding boxes given RGB observations of environments unknown a priori (see, e.g., (Kirillov et al. 2023)).

We assume the preconditions of an action depend only on the state of the objects involved in the action (which is a common assumption when specifying action preconditions in PDDL). We also assume that a lifted action produces the same observable effects regardless of the objects involved in the action. For example, in blocksworld, consider a state with a red and a blue block on a table, then executing a PICKUP action would produce the observation of either a blue or a red block being held, depending on the picked block. For the abovementioned assumption to hold, the PICKUP action is differentiated into PICKUP-RED and PICKUP-BLUE, as picking up blocks of different colors produces different observed effects. Without the above assumption, the same action can have observed effects that are conditioned on the state of the involved objects, i.e., conditional effects. It is worth noting that, an action with conditional effects can be automatically compiled into a set of classical planning actions, one for every conditional effect (Gerevini, Percassi, and Scala 2024); the automated compilation of actions with conditional effects is left as future work.

Finally, we assume objects with similar observations exhibit similar behaviours, thus enabling the action preconditions and effects learned by observing an object to be generalized to different objects with similar observations. For example, in the blocksworld domain, after the preconditions and effects of the action PICKUP-RED($block_0$) have been learned from the observation of $block_0$, the preconditions and effects can be generalized to pickup any red block.

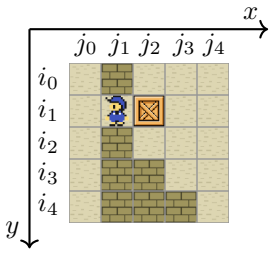


Figure 1: A state image in the Sokoban domain, where an agent is at position $pos_{11}$, and can either push the box right from $pos_{12}$ to $pos_{13}$ or move left to $pos_{10}$. The $x$ and $y$ axes are the reference system of the object bounding boxes.


Figure 2: Observations of some object pairs in the Sokoban domain. In (a) and (c), the agent is below and above a wall, respectively; in (b) the agent is to the left of a box, whereas in (d), it is to the right of a traversable position.

## 4 Method

We describe the proposed solution approach by adopting a running example based on the Sokoban IPC domain (Fern, Khardon, and Tadepalli 2011).

**Example 1 (Sokoban)** *An agent can move between adjacent positions of a grid-based environment, where every position is either traversable, or occupied by a wall (i.e., non-traversable) or a box. The set of actions executable by the agent consists of moving up/down/left/right between adjacent positions, and pushing the box up/down/left/right from a position to an adjacent traversable one, when the agent position is adjacent to the box position.*

An example of subsymbolic state (i.e., RGB image) for the Sokoban domain is shown in Figure 1. The bounding box of an object is defined by a tuple $\langle x_1, y_1, x_2, y_2 \rangle$ of pixel coordinates in the state image, with $\langle x_1, y_1 \rangle$ and $\langle x_2, y_2 \rangle$ being the top-left and bottom-right corners of the bounding box, respectively. Since the agent can recognize object bounding boxes, it can extract from every state the RGB images of all $n$-tuple of objects. We refer to the image of an $n$-tuple of objects as *observation*, and denote by $\mathcal{X}$ the set of observations of all $n$-tuple of objects. For example, the *observation* of an object pair can be obtained by cropping a state RGB image with the minimum/maximum $x$ and $y$ coordinates of the object bounding boxes in the image. Figure 2 shows examples of observations of object pairs in the Sokoban domain.

**Observations of preconditions and effects.** Due to assumptions in Section 3, when the agent executes an action, the action preconditions can be observed just from a portion of the state image, i.e., from the observations of the objects involved in the action. For example, in the Sokoban domain, let $pos_{ij}$ denote the grid position at coordinates $(i, j)$, the action MOVE-LEFT($pos_{11}$, $pos_{10}$) is applicable in a state where the agent is in $pos_{11}$, $pos_{10}$ is adjacent to $pos_{11}$, and $pos_{10}$ is not occupied by a wall or a box; such preconditions are shown in Figure 2d. The same preconditions must hold for any action MOVE-LEFT($pos_{ij}$, $pos_{i'j'}$) with $i, j \neq i', j'$, in environments where the observations of the agent and traversable positions are similar to the ones in Figure 1.

Similarly, the action effects can be observed from a portion of the state image. For example, after executing the action MOVE-LEFT($pos_{11}$, $pos_{10}$), the observations of the object pairs in Figure 2d is no longer observed, and the observation in Figure 2e is observed. However, the action MOVE-LEFT($pos_{11}$, $pos_{10}$) causes the agent to get further from the box placed at position $pos_{12}$ (i.e., the observation in Figure 2b is no longer observed after executing the action), despite $pos_{12}$ is not explicitly mentioned in the action parameters.

A typical assumption when specifying action effects in PDDL is that they can affect only the objects involved in the action. In this work, we drop such an assumption, and learn planning domains where the effects of an action can affect objects not explicitly involved in the action.

The proposed solution approach builds upon the above considerations on observations of preconditions and effects.

### Subsymbolic Learning of Planning Domains

We propose an approach, named S-LAM, for learning an RMDP given a set of execution traces with subsymbolic state images and symbolic actions. S-LAM invents symbolic predicates from the observed $n$-tuple of objects in the trace states; it learns to map the invented predicates into object subsymbolic observations, and vice versa. Afterward, S-LAM learns an RMDP whose transition function is defined over the symbolic state space induced by the invented predicates, and the action space is inferred from the actions executed in the traces. We first describe how the symbolic predicates are invented, and how they are grounded to object subsymbolic observations. Then, we detail how an RMDP can be learned based on the invented predicates and traces.

**Symbolic predicates invention** Firstly, S-LAM extracts from the state RGB images in the input traces the set $\hat{\mathcal{X}} \subseteq \mathcal{X}$ of observations of $n$-tuple of objects, with $n \in \{1, 2\}$. We denote the observation $x \in \mathcal{X}$ of an $n$-tuple of objects $\langle c_1, \ldots, c_n \rangle$ in $\mathcal{C}$ by $x_{\langle c_1, \ldots, c_n \rangle}$; for the sake of presentation, we refer to $x_{\langle c \rangle}$ as $x_c$.

The set $\hat{\mathcal{X}}$ of object observations is then discretized into a set $\mathcal{K}$ of clusters by means of a discretization function $f_k : \mathcal{X} \to \mathcal{K}$. For every $x \in \hat{\mathcal{X}}$, $f_k(x)$ returns the cluster to which $x$ belongs[1]. S-LAM learns $f_k$ by applying an unsupervised clustering algorithm to the observations in $\hat{\mathcal{X}}$. More precisely, for every $n$ in $\{1, 2\}$, S-LAM differentiates the subset $\hat{\mathcal{X}}^{(n)}$ of observations of $n$-tuple of objects in $\hat{\mathcal{X}}$, i.e., $\hat{\mathcal{X}} = \hat{\mathcal{X}}^{(1)} \cup \hat{\mathcal{X}}^{(2)}$ and $\hat{\mathcal{X}}^{(1)} \cap \hat{\mathcal{X}}^{(2)} = \emptyset$. Consequently, the set of clusters is $\mathcal{K} = \mathcal{K}^{(1)} \cup \mathcal{K}^{(2)}$, where a cluster in $\mathcal{K}^{(n)}$ is denoted by $k_i^{(n)}$; similarly, $f_k$ can be factorized into

[1] In this work, we consider discretization functions that are deterministic.

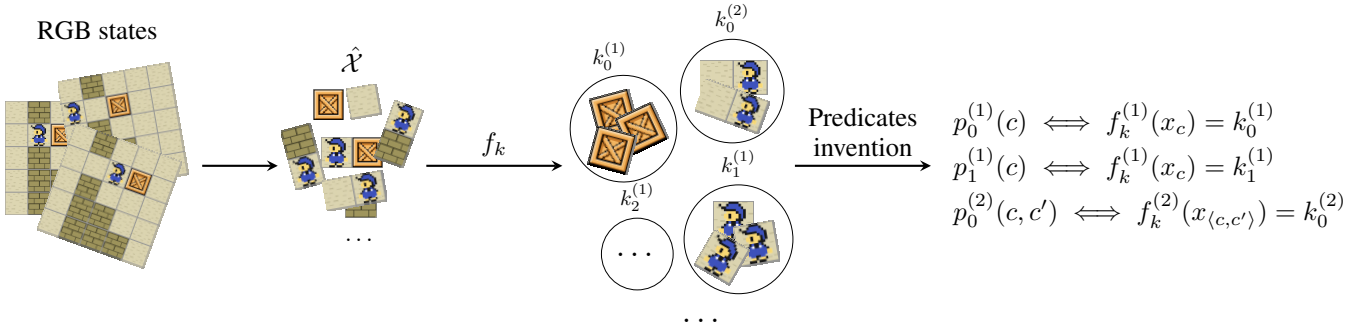


Figure 3: Symbolic predicates invented by S-LAM when learning from Sokoban state images. The set $\hat{\mathcal{X}}$ of object observations extracted from the state images is discretized into a set of clusters, and a symbolic predicate is invented for every cluster.

$\langle f_k^{(1)}, f_k^{(2)} \rangle$, where $f_k^{(n)} : \hat{\mathcal{X}}^{(n)} \rightarrow \mathcal{K}^{(n)}$. For example, in the Sokoban domain, $f_k^{(1)}$ maps the set $\hat{\mathcal{X}}^{(1)}$ of single object images (e.g., images of a player, or a box) into a set $\mathcal{K}^{(1)}$ of clusters.

For every cluster $k_i^{(n)} \in \mathcal{K}^{(n)}$, S-LAM invents a predicate $p_i^{(n)}$ with arity $n$, such that $p_i^{(n)}(c_1, \ldots, c_n)$ is true iff $f_k(x_{\langle c_1,\ldots,c_n \rangle}) = k_i^{(n)}$. We denote the $n$-ary *predicate classifier* function by $\rho^{(n)} : \mathcal{K}^{(n)} \rightarrow \mathcal{P}^{(n)}$, and $\rho = \langle \rho^{(1)}, \ldots, \rho^{(n)} \rangle$; S-LAM considers $\rho^{(n)}$ as a bijective mapping between clusters in $\mathcal{K}^{(n)}$ and $n$-ary predicates in $\mathcal{P}^{(n)}$. For example, in Figure 3, the cluster $k_0^{(1)}$ is produced by the discretization of $\hat{\mathcal{X}}^{(1)}$, and the associated predicate $p_0^{(1)}$ is invented, i.e., $\rho^{(1)}$ maps $k_0^{(1)}$ into $p_0^{(1)}$, and viceversa; for an object $c \in \mathcal{C}$, $p_0^{(1)}(c)$ is true in a state $s$ if the image $x_c$ extracted from $s$ belongs to $k_0^{(1)}$. For example, in Figure 3, $p_0^{(1)}(c)$ represents the fact that $c$ is a box; similarly, $p_0^{(2)}(c, c')$ indicates that player $c'$ (or $c$) is to the right of a traversable position $c$ (or $c'$).

The learned set of predicates $\mathcal{P}$ and discretization function $f_k$ allow to map a state RGB image into a symbolic representation by means of an *encoding* function $enc : \mathcal{X} \rightarrow 2^{\mathcal{P}(\mathcal{C})}$, composed as $enc = (\rho \circ f_k)$, which is applied on the set $\hat{\mathcal{X}}$ of object observations extracted from the state image.

Similarly, for every $n$-tuple of objects in $\hat{\mathcal{C}} \subseteq \mathcal{C}$, S-LAM learns to map symbolic states from $2^{\mathcal{P}(\hat{\mathcal{C}})}$ into object observations in $\mathcal{X}^{(n)}$ by means of a *decoding* function $dec : 2^{\mathcal{P}(\mathcal{C})} \rightarrow \mathcal{X}$. For example, when $\hat{\mathcal{C}} = \{c\}$, the decoding function firstly maps a true predicate $p_i^{(1)}(c)$ into its corresponding cluster $k_i^{(1)}$, and then sample from $k_i^{(1)}$ an observation $x_c \in \mathcal{X}^{(1)}$. Notice that, S-LAM assume predicates in $\mathcal{P}^{(n)}$ to be mutually exclusive, i.e., if $p_i^{(n)}(c)$ is true in a state, then $p_j^{(n)}(c)$ is false in the state, for every $p_j^{(n)} \in \mathcal{P}^{(n)}, p_j^{(n)} \neq p_i^{(n)}$.

**Learning an RMDP.** S-LAM learns an RMDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$ from a set of traces, where $\mathcal{S} = 2^{\mathcal{P}(\mathcal{C})}$ is the state space induced by the invented set of predicates $\mathcal{P}$, and constants $\mathcal{C}$ in the traces; $\mathcal{A}$ is a discrete and symbolic action space inferred from the actions observed in the traces, where every action is assumed to be applicable to any object (i.e., the objects share the same type); $\mathcal{R}$ and $\gamma$ are defined later when describing how to plan with the learned RMDP; $\mathcal{T}$ is the transition function derived as detailed in the following.

S-LAM learns $\mathcal{M}$ in two stages: firstly, it maps the trace subsymbolic states into relational ones, from which it learns *lifted* action schemas. Secondly, it derives $\mathcal{T}$ by combining the previously learned encoding/decoding functions and lifted action schemas.

To learn the action schemas, S-LAM extracts from each state image the set of object observations $\hat{\mathcal{X}}$, and derives the relational state $s = enc(\hat{\mathcal{X}})$ associated with the image (which is represented in terms of the previously invented set of predicates $\mathcal{P}$). With the resulting traces, S-LAM can take advantage of an off-the-shelf Action Model Learning (AML) approach for learning lifted action schemas from symbolic traces with noisy states; see, e.g., (Mourão et al. 2012; Segura-Muros, Pérez, and Fernández-Olivares 2018; Lamanna and Serafini 2024). An example of the abovementioned procedure is represented in Figure 4, for an execution trace composed of a single transition in the Sokoban domain.

Notice that, the learned action schemas cannot be directly used for solving planning problems. Consider the schema learned in Figure 4 for the lifted action MOVE-RIGHT($v_1$, $v_2$). After executing MOVE-RIGHT($pos_{10}$, $pos_{11}$) in the transition in Figure 4, the ground atom $p_1^{(2)}(pos_{01}, pos_{11})$ becomes false, and $p_2^{(2)}(pos_{01}, pos_{11})$ becomes true; this is the case when, e.g., $p_2^{(2)}$ is associated with the cluster containing images as the one in Figure 2a. However, the object $pos_{01}$ is not involved in the action MOVE-RIGHT($pos_{10}$, $pos_{11}$); therefore, such an effect cannot be modelled in the PDDL specification of the action schema shown in Figure 4. A possible solution might be modifying the action parameters to include all objects that can possibly be affected by the action effects. In this work, we adopt an alternative approach by representing the planning domain as an RMDP, which allows to express the previously described effects in the transition function. To this aim, we assume that the properties of the objects not involved in an action do not change after exe-

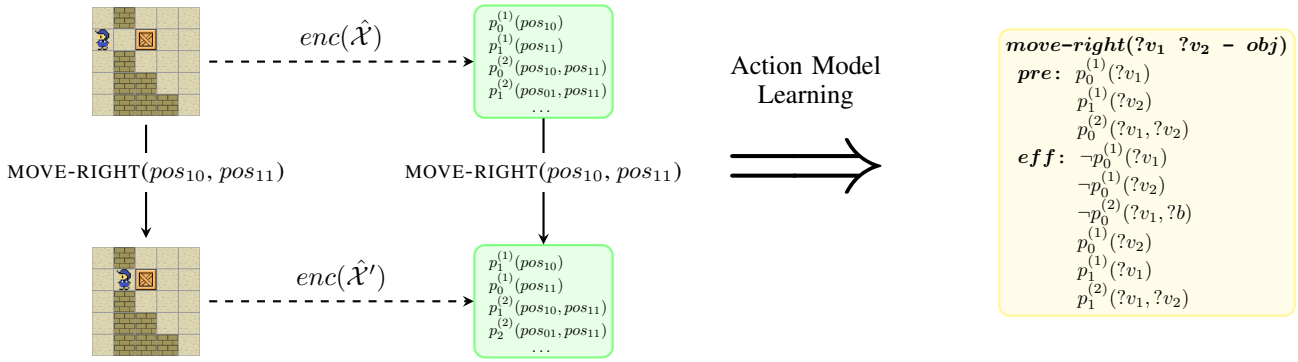


Figure 4: Example of learning lifted preconditions and effects from a single transition of action MOVE-RIGHT in the Sokoban domain. S-LAM extracts the set of images $\hat{\mathcal{X}}$ and $\hat{\mathcal{X}}'$ from the state before and after executing the action, respectively. Afterwards, it maps the object observations into symbolic representations by means of the encoding function $enc$; finally, it learns the lifted action schema of MOVE-RIGHT by applying an off-the-shelf AML approach.

cuting the action, while we admit the situation where the relations between such objects and the action objects change. For example, in Figure 4, we admit the relation between the player and a wall changes after moving right (even though the wall object is not explicitly involved in the action), while we assume the properties of the walls do not change.

To describe how the transition function $\mathcal{T}$ is modelled, we refer to the example in Figure 5. We aim to predict the relational state $s' \in \mathcal{S}$ reached after executing an action $a \in \mathcal{A}$ in $s \in \mathcal{S}$, given the state image before executing the action. S-LAM extracts from the state image the set of object observations $\hat{\mathcal{X}}$, and encode the observations into the relational state $s = enc(\hat{\mathcal{X}})$. Next, it predicts the state of the set $\hat{\mathcal{C}} \subseteq \mathcal{C}$ of objects involved in the action, i.e. the truth values of the atoms in $\mathcal{P}(\hat{\mathcal{C}})$, by means of the (previously learned) schema of $a$. Next, the predicted state in $2^{\mathcal{P}(\hat{\mathcal{C}})}$ is decoded using the decoding function $dec$, and the decoded object observations are composed with the previous state image to derive the next state image. This is achieved by replacing the observation of every object $c \in \hat{\mathcal{C}}$ in the previous state image with the observation obtained by decoding the true ground atom $p_i^{(1)}(c) \in \mathcal{P}^{(1)}(\hat{\mathcal{C}})$ for some $i$. While more sophisticated generative models (e.g., (Zhang et al. 2024)) could be applied for generating the next state image, in our experiments this method was effective enough to model the transition function in the experimented environments. Finally, the next state $s'$ is obtained by encoding the objects observations in the next state image by means of $enc$. Figure 5 shows an example of prediction of the next state $s'$ for the same transition considered in Figure 4 in the Sokoban domain, where we omitted the final decoding step to derive $s'$ from the object observations in the predicted state image.

### Planning with the Learned RMDP

To plan with the learned RMDPs, we adopt the Upper Confidence bounds for Trees (UCT), a Monte-Carlo Tree Search based online planning algorithm (Silver and Veness 2010). The set of goal states of a planning problem is provided to the agent in terms of goal state (RGB) images. Given a number $g$ of goal images, where $\hat{\mathcal{X}}_i$ is the set of object observations derived from the $i$-th goal image, the set $\mathcal{G} \subseteq \mathcal{S}$ of goal states in the agent RMDP is $\mathcal{G} = \{enc(\hat{\mathcal{X}}_i)\}_{i=1}^{g}$.

For every RMDP state $s \in \mathcal{S}$, the distance $dist(s, \mathcal{G})$ between $s$ and $\mathcal{G}$ is heuristically estimated by considering the minimum set $\mathcal{C}_g \subseteq \mathcal{C}$ of objects involved by ground atoms that are true in a state $s_g \in \mathcal{G}$ but not in $s$. For the objects in $\mathcal{C}_g$, $dist(s, \mathcal{G})$ sums the pairwise distances of the bounding box centroids in the images of $s_g$ and $s$. With the set $\mathcal{G}$ of goal states, the reward function for every triple $\langle s, a, s' \rangle \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is defined as $\mathcal{R}(s, a, s') = -dist(s', G) - |\mathcal{C}_g| * \lambda$, where $\lambda \in \mathbb{R}$.

## 5 Experiments

We experimented with S-LAM in 4 IPC domain variants: Sokoban, Visitall, Floortile, and a simplified version of Sokoban, referred to as Gridnav. In all considered domains, an agent navigates between adjacent locations in a grid-based environment, where some locations are not traversable. The Sokoban domain is described in Example 1, whereas Gridnav is a simplification of Sokoban where there are no boxes, and the agent's goal is to navigate to a goal position. In the Visitall domain, an agent is required to visit a set of goal locations. In Floortile, there are (possibly) multiple agents, each equipped with a color that is either white or black, and the agents are tasked to paint some goal locations, where every location is no longer traversable after being painted. For every domain, we implemented a simulator with grid-based RGB images of the environment states, an example image for every domain is reported in Figure 6.

As execution traces for learning the RMDPs of every domain, we adopted the set of 10 traces provided in (Stern et al. 2025). The traces have an average number of 25.4 actions for domains Gridnav and Sokoban, 7.9 for Visitall, and 34.9 for Floortile, and have been generated in grids of sizes ranging from $3 \times 3$ to $10 \times 10$. We modified both the domains and the trace actions to respect the assumptions described in Section 3. For example, in

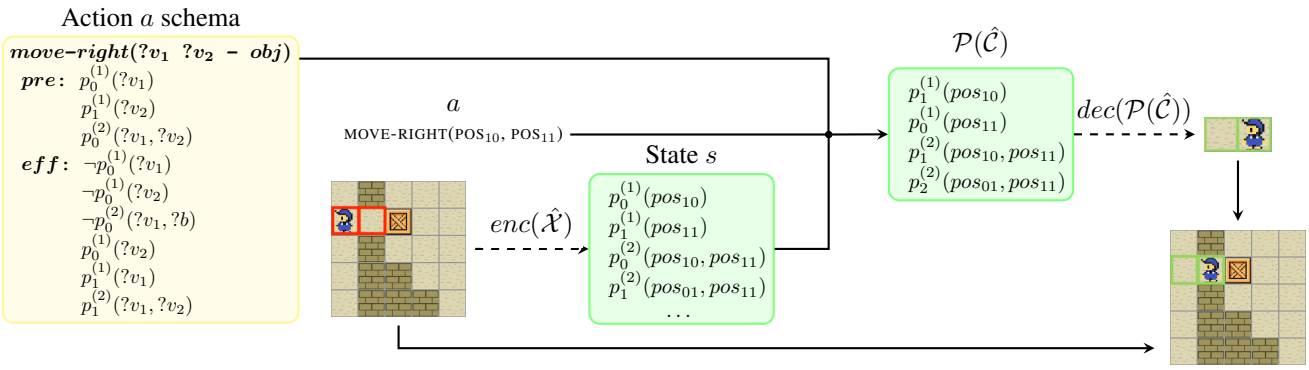


Figure 5: Prediction of the state $s'$ reached after executing an action $a$ in a state $s$ in the RMDP learned by S-LAM.

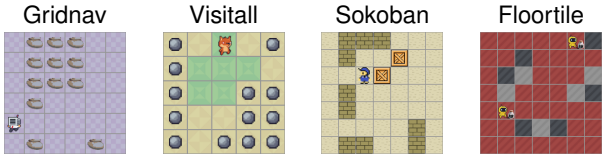


Figure 6: Example of state RGB images for the IPC domain variants adopted in the experiments

Sokoban, the parameter $direction$ is removed from the IPC domain action MOVE($pos_x, pos_y, direction$) by compiling the action into 4 actions: MOVE-LEFT($pos_x, pos_y$), MOVE-RIGHT($pos_x, pos_y$), MOVE-UP($pos_x, pos_y$), and MOVE-DOWN($pos_x, pos_y$); similarly, we modified the actions in Sokoban traces[2].

We investigate the planning efficacy of the RMDPs learned by S-LAM by planning in 10 test problems provided by (Stern et al. 2025) to evaluate domain model learning approaches. It is worth noting that the test problems are different from the ones from which the traces were generated. The problems correspond to environments with grid sizes ranging from $5\times5$ to $10\times10$[3]. For every problem, we run UCT with the RMDPs learned by S-LAM for a maximum number of 100 steps, we call each of these run an episode. An episode ends if a goal state is reached, or the maximum number of execution steps is executed. To specify the set of goal states in the learned RMDPs, S-LAM is provided with the set of goal images, which are encoded into the RMDP goal states by means of the encoding function $enc$. Since the set of goal states can be large in complex problems (e.g., thousands of images), we randomly sampled a maximum number of 300 goal images for each problem.

**Experimental setting.** To cluster the object observations extracted from state images, we used the Birch algorithm (Zhang, Ramakrishnan, and Livny 1997), which does not require an input number of clusters. Since the number of object pairs can become very large in problems with a high number of objects, S-LAM considers only the pair of objects whose centroid distance is lower than a given threshold (set to 50 in our experiments).

To learn the lifted action schemas from noisy symbolic traces, we adopted the approach proposed in (Lamanna and Serafini 2024), with a noise rate set to 0.05. For online planning with the UCT algorithm, the tree search depth is set to 20 and 200 actions are simulated at each planning step, with an exploration constant equal to $10^2$. For the reward function, $\lambda = 10^2$ and $\gamma = 0.95$. All experiments were run on an Apple M1 Pro CPU with 16 GB of RAM.

**Evaluation metrics.** We evaluate the planning efficacy with the learned RMDPs by means of standard evaluation metrics adopted in Embodied AI (Anderson et al. 2018). The *Success Rate* (*SR*) is a binary value equal to 1 when the episode succeeds, i.e., if the agent is in a goal state when the episode ends. The *Distance To Success* (*DTS*) is the length of the optimal plan for achieving the goal from the state at the end of the episode, where $DTS > 0$ only when $SR = 0$. The *Success weighted by Path Length* (*SPL*) measures the optimality of the plan $\pi$ executed by the agent w.r.t. the optimal plan $\pi^\star$, i.e., $SPL = SR\frac{|\pi^\star|}{\max(|\pi|,|\pi^\star|)}$.

In our experimental analysis, we evaluate a more fine-grained variant of the *SR*, which measures the *goal achievement* regardless of the agent being in a goal state at the end of an episode. The goal achievement $G \in [0, 1]$ is the maximum ratio of ground atoms that are true in both a goal and agent state at the end of an episode. Formally, for a given set $\mathcal{G} \subseteq \mathcal{S}$ of goal states and state $s \in \mathcal{S}$, the goal achievement in $s$ is $G = \max_{s_g \in \mathcal{G}}(1 - \frac{|s_g \setminus s|}{|s_g|})$, where $|s_g \setminus s|$ denotes the size of ground atoms that are true in $s_g$ and false in $s$.

**Experimental results** We compare the performance achieved by planning with the RMDPs learned by S-LAM with two baselines: an agent that executes random actions, referred to as Random, and an agent named Greedy that executes actions maximising the next state reward, and randomly breaking ties. In Figure 7, we report the cumulative average of $G$ and *SPL* over all episodes.

The average $G$ achieved by S-LAM is higher than com-

[2]The PDDL domains and traces are provided in the supplementary material.

[3]For further details on the problems and traces, we refer to (Stern et al. 2025)

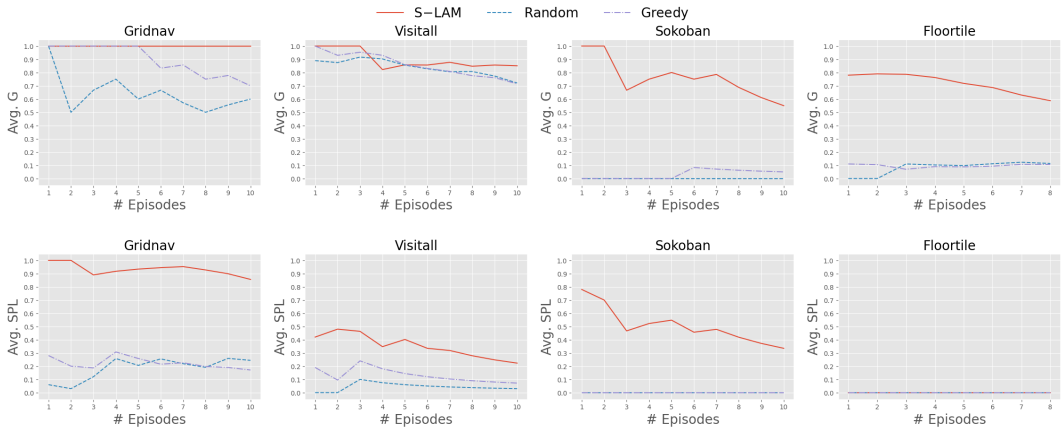


Figure 7: Goal achievement and *SPL* achieved by S-LAM, Random, and Greedy in every IPC variant domain. The metric values are the cumulative average over the number of episodes.

pared baselines in all episodes but one in the Visitall domain, while the *SPL* is consistently higher for S-LAM than compared baselines in all domains but Floortile. In Gridnav, S-LAM solves all problems by producing near-optimal plans (i.e., average $SPL > 0.85$). The Random and Greedy baselines achieve competitive values of $G$ in Gridnav, indicating that the problems are not difficult to solve. Notably, neither Random nor Greedy is able to solve any problem in Sokoban, where S-LAM outperforms both baselines in terms of $G$ and *SPL* by a large margin; similarly for the goal achievement in Floortile. However, in Floortile, the *SPL* equals 0 for all approaches, as no approach achieves $G = 1$ in any problem, indicating that, in our experiments, the planning problems are more complex for Floortile than for other domains. The average *DTS* achieved by S-LAM in Gridnav and Visitall is 0 and 11.1, respectively; for Random (resp. Greedy), the *DTS* equals 1.8 (resp. 3) in Gridnav, and 14.7 (resp. 16.1) in Visitall. We do not report the *DTS* for Sokoban and Floortile, as these domains have non reversible actions, which make the problem infeasible in some episodes, and thus prevent from evaluating the *DTS*.

Overall, the performance achieved by Greedy is better than or comparable with Random, indicating that the reward function can provide effective guidance. Interestingly, the performance gap between S-LAM and Greedy provides empirical evidence of the effectiveness of planning with the learned RMDPs. Learning the RMDPs using S-LAM required 20.26 CPU time seconds on average for all domains. The CPU time seconds required at each step by UCT for planning with the learned RMDPs are 0.65 for Gridnav, 0.91 for Visitall, 1.19 for Sokoban, and 37.73 for Floortile. We believe such a large time for Floortile is due to the high number of (22) lifted actions in the domain variant. Due to Floortile complexity, the experiments in the two most complex problems ran out of memory and are not included in Figure 7.

The failures of S-LAM are either due to the reward signal not being sufficiently effective, which is likely to be the case in large grids, or to the agent executing non reversible actions that make the problem infeasible. For example, in Sokoban 40% of the failures are due to non reversible actions and 60% to the reward signal inefficacy. To mitigate the reward effectiveness, a possible approach can be increasing the depth of the search tree used by UCT, and the number of simulated actions, at the cost of increasing the CPU time required for planning.

## 6 Conclusions and Future Work

This paper considers the problem of offline learning planning domains from execution traces with subsymbolic observations of states (i.e., RGB images), and symbolic actions. The proposed approach (S-LAM) does not assume the set of domain predicates to be given as input, and invents symbolic predicates that are grounded to the observed effects in the traces. The learned planning domains are modeled as lifted RMDPs defined over the state space induced by the learned predicates.

While S-LAM provides advances over state-of-the-art approaches that learn planning domains from subsymbolic observations, it has some limitations given by the adopted assumptions. Firstly, the considered environments are deterministic and fully observable, though in principle, the learned RMDPs allow for modeling probabilistic transitions. Secondly, different trace actions are assumed to produce different observable effects, while in real-world environments, the observed effects of an action (e.g., picking up a block) may depend on the state of the objects involved in the action (e.g., the color of the picked block). A possible solution can be compiling each conditional observed effect into a different action, similarly to (Gerevini, Percassi, and Scala 2024). Finally, the experimented domains provide grid-based RGB images, and the observed objects are grid cells with fixed positions, which does not require to consider object tracking, and make the prediction of future state images easier during planning. More complex RGB observations could be considered by adopting more sophisticated object tracking and generative models. Lifting the above assumptions is part of future work.

## References


Aineto, D.; Celorrio, S. J.; and Onaindia, E. 2019. Learning action models with minimal observability. *Artificial Intelligence*, 275: 104–137.

Aineto, D.; and Scala, E. 2024. Action Model Learning with Guarantees. *arXiv preprint arXiv:2404.09631*.

Anderson, P.; Chang, A.; Chaplot, D. S.; Dosovitskiy, A.; Gupta, S.; Koltun, V.; Kosecka, J.; Malik, J.; Mottaghi, R.; Savva, M.; et al. 2018. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*.

Asai, M. 2019. Unsupervised grounding of plannable first-order logic representation from images. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling*, volume 29, 583–591.

Asai, M.; Kajino, H.; Fukunaga, A.; and Muise, C. 2022. Classical planning in deep latent space. *Journal of Artificial Intelligence Research*, 74: 1599–1686.

Asai, M.; and Muise, C. 2020. Learning Neural-Symbolic Descriptive Planning Models via Cube-Space Priors: The Voyage Home (to STRIPS). In *the International Joint Conference on Artificial Intelligence (IJCAI)*, 2676–2682.

Callanan, E.; Venezia, R. D.; Armstrong, V.; Paredes, A.; Chakraborti, T.; and Muise, C. 2022. MACQ: A Holistic View of Model Acquisition Techniques. In *ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.

Fern, A.; Khardon, R.; and Tadepalli, P. 2011. The first learning track of the international planning competition. *Machine Learning*, 84(1-2): 81–107.

Gerevini, A. E.; Percassi, F.; and Scala, E. 2024. An effective polynomial technique for compiling conditional effects away. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 20104–20112.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated planning: theory & practice*. Elsevier.

Gösgens, J.; Jansen, N.; and Geffner, H. 2025. Learning lifted strips models from action traces alone: A simple, general, and scalable solution. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 35, 189–197.

Kirillov, A.; Mintun, E.; Ravi, N.; Mao, H.; Rolland, C.; Gustafson, L.; Xiao, T.; Whitehead, S.; Berg, A. C.; Lo, W.-Y.; et al. 2023. Segment anything. In *Proceedings of the IEEE/CVF international conference on computer vision*, 4015–4026.

Konidaris, G.; Kaelbling, L. P.; and Lozano-Pérez, T. 2018. From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning. *Journal of Artificial Intelligence Research*, 61: 215–289.

Lamanna, L.; and Serafini, L. 2024. Action Model Learning from Noisy Traces: a Probabilistic Approach. In *ICAPS*, 342–350.

Lamanna, L.; Serafini, L.; Saetti, A.; Gerevini, A. E.; and Traverso, P. 2025. Lifted action models learning from partial traces. *Artificial Intelligence*, 339.

Le, H. S.; Juba, B.; and Stern, R. 2024. Learning Safe Action Models with Partial Observability. In *AAAI Conference on Artificial Intelligence*, 20159–20167.

Mordoch, A.; Juba, B.; and Stern, R. 2023. Learning Safe Numeric Action Models. In *AAAI Conference on Artificial Intelligence*, 12079–12086.

Mordoch, A.; Scala, E.; Stern, R.; and Juba, B. 2024. Safe learning of pddl domains with conditional effects. In *International Conference on Automated Planning and Scheduling*, volume 34, 387–395.

Mourão, K.; Zettlemoyer, L.; Petrick, R. P.; and Steedman, M. 2012. Learning STRIPS operators from noisy and incomplete observations. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, 614–623.

Pu, Y.; Gan, Z.; Henao, R.; Yuan, X.; Li, C.; Stevens, A.; and Carin, L. 2016. Variational autoencoder for deep learning of images, labels and captions. *Advances in neural information processing systems*, 29.

Segura-Muros, J. A.; Pérez, R.; and Fernández-Olivares, J. 2018. Learning numerical action models from noisy and partially observable states by means of inductive rule learning techniques. *KEPS 2018*, 46.

Silver, D.; and Veness, J. 2010. Monte-Carlo planning in large POMDPs. *Advances in neural information processing systems*, 23.

Stern, R.; and Juba, B. 2017. Efficient, Safe, and Probably Approximately Complete Learning of Action Models. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 4405–4411.

Stern, R.; Lamanna, L.; Mordoch, A.; Benyamin, Y.; Lauer, P.; Juba, B.; Behnke, G.; Muise, C.; Bercher, P.; Vallati, M.; et al. 2025. Evaluating Planning Model-Learning Algorithms.

van Otterlo, M. 2012. *Solving Relational and First-Order Logical Markov Decision Processes: A Survey*, 253–292. Berlin, Heidelberg: Springer Berlin Heidelberg.

Wu, K.; Yang, Q.; and Jiang, Y. 2007. ARMS: An automatic knowledge engineering tool for learning action models for AI planning. *The Knowledge Engineering Review*, 22(2): 135–152.

Xi, K.; Gould, S.; and Thiébaux, S. 2024. Neuro-Symbolic Learning of Lifted Action Models from Visual Traces. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 34, 653–662.

Younes, H. L. S.; and Littman, M. L. 2004. PPDDL 1.0 : An Extension to PDDL for Expressing Planning Domains with Probabilistic Effects. Technical report, School of Computer Science, Carnegie Mellon University. Techn. Rep. CMU-CS-04-162.

Zhang, J.; Huang, J.; Jin, S.; and Lu, S. 2024. Vision-language models for vision tasks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 46(8): 5625–5644.

Zhang, T.; Ramakrishnan, R.; and Livny, M. 1997. BIRCH: A new data clustering algorithm and its applications. *Data mining and knowledge discovery*, 1(2): 141–182.