Learning Lifted Action Models from Traces with Minimal Information About Actions and States

Jonas Gösgens, Niklas Jansen, Hector Geffner

RWTH Aachen University

{jonas.goesgens, niklas.jansen, hector.geffner}@ml.rwth-aachen.

Abstract

It has been recently shown that lifted STRIPS models can be learned correctly and efficiently from action traces alone; i.e., applicable action sequences from a hidden STRIPS model. The result is remarkable because the states are not assumed to be observable at all, and yet it is not practical enough as STRIPS actions include arguments that are not needed for selecting the actions. This shortcoming has been addressed by assuming that the action traces come instead from a hidden STRIPS+ model where some action arguments are implicit in the hidden action preconditions. A limitation of this approach, however, is that it assumes that the states are fully observable. In this work, we relax these restrictions and consider the problem of learning STRIPS+ action domains from traces in a more general context where the traces carry partial information about both actions and states. In particular, we formulate algorithms and completeness results for three general cases, all of which assume full observability of selected action arguments. In the first case, no observability of the state is assumed; in the second case, full observability of some state predicates is assumed, and in the third case, local observability of some state predicates is assumed instead. Given a STRIPS+ domain, these results characterize the conditions under which an equivalent domain can be learned from traces. Experimental results are reported.

Introduction

Recently, it has been shown that lifted STRIPS models can be learned correctly and efficiently from applicable action sequences (action traces) drawn from a hidden STRIPS model, via a simple algorithm called SIFT (Gösgens, Jansen, and Geffner 2025). The result, which builds on earlier work (Cresswell, McCluskey, and West 2013; Gregory and Cresswell 2015), is particularly interesting as it implies that the domain predicates can be learned accurately and effectively from action traces alone, without any restrictions, and without assuming state observability.[1]

Still, the result is not practical enough, and it does not provide a *lifted alternative to model-based reinforcement learning* approaches that are also aimed at learning dynamic models from traces (Sutton and Barto 2018). The reason is that STRIPS action traces are not natural and often presume knowledge of the hidden domain that is to be learned. For example, in order to learn a domain like the sliding-tile puzzles, the STRIPS actions take as arguments the tile to be moved, its current position, and the next position. Similarly, in the Blocksworld, an action like $unstack(x, y)$ has to mention not only the block to be unstacked but also the block that is below it. Many of these action arguments, however, are not needed for selecting and identifying the action to be done, but for modeling the action in STRIPS.

The problem of learning action models in more expressive action languages that do not require as many parameters has not received as much attention, and a recent approach moves away from STRIPS to a simple extension called STRIPS+, where action arguments can be conveyed explicitly, as usual, or implicitly, as variables that bind to unique objects in the action preconditions (Jansen, Gösgens, and Geffner 2025). Indeed, the first version of PDDL accommodated such *implicit action arguments* through the :VARS keyword (McDermott et al. 1998). In STRIPS+, the action of sliding a tile to the left can be expressed indeed without any arguments. The precondition containing the lifted atoms $\mathrm{atB}(z_1)$, $\mathrm{leftof}(z_2, z_1)$, and $\mathrm{atT}(z_3, z_2)$, encoding the position of the blank, the corresponding adjacency relation, and the tile to be moved and its position, defines the action arguments implicitly by the unique objects that can bind to the z_i variables in a state.

A limitation of the resulting algorithm, called SYNTH (Jansen, Gösgens, and Geffner 2025), is that it assumes that the states are *fully observable*, which means that all the predicates are given, and none has to be "invented". In a way, the SIFT algorithm "invents" predicates from full STRIPS action traces, while SYNTH deals with less informed STRIPS+ action traces but assumes that the predicates are given. In this work, we build on these approaches and algorithms to investigate what partial information is needed on both actions and states in the traces in order to learn a domain. In particular, we formulate algorithms and completeness results for three general cases, all of which assume full observability of the STRIPS+ actions in the traces. In the first case, no observability of the state is assumed; in the second case, full observability of some state predicates is assumed; and in the third case, local observability of some state predicates is assumed instead. Given a STRIPS+ domain, one can then

[1] The only assumption is that the actions are well-formed, meaning that they do not add atoms that are already true, or delete atoms that are already false.

determine exact conditions under which an equivalent domain will be learned from traces. For example, the STRIPS+ domain for the sliding-tile puzzle with no action arguments can be learned provided that the atB predicate encoding the blank position is *fully observed*, and the atT relation encoding the tile position is *locally observable.* The resulting domain learning algorithm, called SYNTH+, is an extension and generalization of both SIFT and SYNTH.

The results are different from those that allow some observations in STRIPS traces to be missing or corrupted by noise. In our case, a non-observed predicate is never observed, and a locally observed predicate defines precisely the observed atoms. From these crisp definitions, we will establish the scope of the model learning algorithms and establish their correctness and completeness conditions.

The paper is organized as follows. We review next related work and relevant background, then we introduce the new notions, the new algorithms, their properties, and report experiments.

Preview

In the Delivery domain, there are packages spread in a grid that an agent must deliver to some target cell. In STRIPS, the domain can be modeled with three action schemas

$$\mathit{move}(c, c'),\ \mathit{pick}(o, c),\ \mathit{drop}(o, c)\ , \tag{1}$$

the first representing the agent moving from a cell to an adjacent cell, the second, picking up an object from a cell, and the third, dropping it on a cell. The SIFT algorithm can learn an equivalent STRIPS model from action traces made of ground instances of these three schemas. In this paper, we introduce algorithms and the corresponding theorems for learning from a broader class of traces. For example, without assuming that the state is observable, the SIFT+ algorithm will learn from STRIPS+ traces from instances of the actions

$$\mathit{move}(c'),\ \mathit{pick}(o),\ \mathit{drop}()\ , \tag{2}$$

where some of the STRIPS action arguments that are not needed for selecting the actions are omitted. This will be achieved by learning the "mutex" predicate $\mathrm{at}(c)$ first, true for a single cell c in any state, encoding the agent position, and then learning from it, the "mutex" predicates $\mathrm{atP}(o, c)$, true for a single cell c per object, and $\mathrm{hold}(o)$, true for a single object o at most. At the same time, SYNTH+ will learn from ground traces of action schemas like

$$\mathit{right}(),\ \mathit{up}(),\ \ldots,\ \mathit{pick}(o),\ \mathit{drop}()\ . \tag{3}$$

provided that the predicate $at(c)$ is fully observed, and that the two directional adjacency relations (e.g., leftof and $\mathrm{belowof}$) are observed. The two other relations needed, $\mathrm{atP}(o, c)$ and $\mathrm{hold}(o)$, will be learned. Finally, in RL, it is common to consider actions with no arguments at all like (Chevalier-Boisvert et al. 2019)

$$\mathit{right}(),\ \mathit{up}(),\ \ldots,\ \mathit{pick}(),\ \mathit{drop}()\ . \tag{4}$$

The corresponding action models will be learned in this case provided that, in addition, there cannot be more than one object in a cell, and that the predicate $\mathrm{atP}(o, c)$ is *locally observed*, meaning that in this case only the true $\mathrm{atP}(o, c)$ atoms for the cell c for which $\mathrm{at}(c)$ is true will be observed. The predicate $\mathrm{hold}(o)$ will be learned.

While neither SIFT nor SYNTH can learn the action models in the last three scenarios, SIFT+ will learn the correct model in the first of them, and SYNTH+, in all of them.

Related Work

While many languages have been developed for representing lifted dynamic models in logical form, the work on learning these models has been focused mostly on STRIPS.

Learning models from actions. The LOCM system (Cresswell and Gregory 2011; Cresswell, McCluskey, and West 2013) accepts action traces as inputs, and outputs lifted domain descriptions, but it is a heuristic algorithm and its scope is not clear. The SIFT algorithm uses full STRIPS action traces and has been shown to be sound, complete, and scalable (Gösgens, Jansen, and Geffner 2025). A SAT-based approach to lifted model learning has been developed as well, but the approach is not scalable (Bonet and Geffner 2020; Rodriguez et al. 2021).

Learning models from states and actions. The problem of learning lifted STRIPS models from state-action traces has received more attention (Zhuo and Kambhampati 2013; Aineto, Celorrio, and Onaindia 2019; Lamanna et al. 2021; Verma, Marpally, and Srivastava 2021; Callanan et al. 2022; Le, Juba, and Stern 2024; Bachor and Behnke 2024; Xi, Gould, and Thiébaux 2024; Aineto and Scala 2024). While observability of the states can be partial or noisy, in almost all cases the observations reveal all the domain predicates and their arities, and the STRIPS actions reveal all the arguments. In contrast, Balyo et al. (2024) introduce a SAT-based learning formulation where only the action names are observed, while Lamanna et al. (2025) deals with partially observable states and actions although it is not complete, as it cannot learn preconditions over action arguments that are not observable and do not appear in the effects. The SYNTH algorithm (Jansen, Gösgens, and Geffner 2025), that learns STRIPS models from STRIPS+ traces combining actions and states, accounts for such missing action arguments using preconditions with free variables.

Model-based RL. Model-based reinforcement learning algorithms learn controllers by also learning (stochastic) models, without making assumptions about the structure of them (Sutton and Barto 2018). In the tabular setting, they result in flat state models with state transition probabilities obtained from simple counts (Brafman and Tennenholtz 2003). In some cases, a first-order state language is assumed but the state predicates are given (Diuk, Cohen, and Littman 2008; Zettlemoyer, Pasula, and Kaelbling 2005). In more recent approaches, the learned dynamics is not represented compactly in languages such as STRIPS or PDDL, but in terms of deep neural networks (Micheli, Alonso, and Fleuret 2023; Hafner et al. 2021; Burchi and Timofte 2025). A limitation

of these methods, like other recent deep-learning approaches that learn STRIPS models from state images (Asai and Fukunaga 2018; Asai et al. 2022), is that the learned action models are opaque and propositional.

Background

We review STRIPS, STRIPS+, and the input traces.

STRIPS

A classical STRIPS problem is a pair $P = \langle \mathcal{D}, I \rangle$ where $\mathcal{D}$ is a first-order *domain* and I contains information about the instance (Geffner and Bonet 2013; Ghallab, Nau, and Traverso 2016). The domain $\mathcal{D} = \langle \mathcal{P}, \mathcal{A} \rangle$ has a set $\mathcal{P}$ of predicate symbols p and a set of action schemas $\mathcal{A}$ with preconditions and effects given in terms of atoms $p(x_1, \ldots, x_k)$, where p is a predicate symbol of arity k, and each x_i is an argument of the schema. The instance information is a tuple $I = \langle O, \mathit{Init}, G \rangle$ where O is a set of object names c_i, and *Init* and G are sets of *ground atoms* $p(c_1, \ldots, c_k)$ denoting the initial and goal situations. A STRIPS problem $P = \langle \mathcal{D}, I \rangle$ defines a state graph $G(P)$ where nodes represent reachable states in P, the root node represents the initial state, and edges indicate state transitions labeled with the actions causing them. A path in this graph represents an action sequence that is applicable in the state represented by the first node. SIFT learns domains expressed in STRIPS with *negation* where negative literals can be used in the initial situation, action preconditions, and goals. The states in such a case are not sets of ground atoms but sets of ground literals. Since the goals of an instance $P = \langle \mathcal{D}, I \rangle$ play no role in learning, we will regard I as just representing the initial situation.

STRIPS+

In order to learn from more natural traces than those resulting from STRIPS models, the SYNTH algorithm learns from a slightly more expressive language called STRIPS+. The main difference between STRIPS+ and STRIPS is that the former accommodates free z_i variables in the action preconditions that must bind uniquely to single objects in each state. Formally, action schemas $a(x)$ in STRIPS+ have (conjunctive) preconditions $\mathrm{Pre}(a(x)) = \Phi(x, y, z)$ with free variables among those of x, y, and z which are pairwise disjoint sets of variables. The x_i and z_i variables in x and z can appear in action effects, but the variables y_i in y cannot. A ground STRIPS+ action $a(o)$ is *applicable* in a state s if its precondition formula $\Phi(x, y, z)$ is *satisfiable* in s with a grounding that binds x to o. A grounding of $\Phi(x, y, z)$ in s is an assignment σ of variables in the formula to constants (objects) in the instance. A grounding *satisfies* $\Phi(x, y, z)$ in s if the resulting ground atoms are all true in s, and the formula $\Phi(x, y, z)$ is *satisfiable* in s if some grounding of the variables satisfies it. The z variables are *determined* by the variables x if the groundings that satisfy the formula must agree on the value (grounding) of z when they agree on the value of x. The *semantics* of the STRIPS+ action $a(x)$ with precondition $\Phi(x, y, z)$ is the semantics of the STRIPS action $a'(x')$ that has the same preconditions and effects as $a(x)$ but with the y and z variables pushed as explicit arguments in x'. Indeed, this is the way to map STRIPS+ models into STRIPS models. As an example, the action $right()$ from Delivery in (3) can be modelled with a precondition $\Phi(x, y, z) \; : \; at(z_1) \wedge rightof(z_1, z_2)$. This precondition uniquely determines the implicit arguments z_1 and z_2 of the action, since the agent is located in exactly one cell and there is only one cell to its right.

Traces

An *action trace* $a_0, \ldots, a_n$ in a domain instance $P = \langle \mathcal{D}, I \rangle$ is an action sequence that is applicable from a reachable state in P. An action trace from $\mathcal{D}$ is a trace from an instance of $\mathcal{D}$. For each trace, there is an initial state s_0 and states s_{i+1} generated by the actions in the trace. The trace $s_0, a_0, s_1, a_1, \ldots, a_{n-1}, s_n$ is called a state-action trace. In this work we consider action traces and partially observable state traces over (hidden) STRIPS+ domains of the form $\omega_0, a_0, \omega_1, a_1, \ldots$ where ω_i is a partial observation of the hidden state s_i.

When learning from action traces or partially observable traces, there is no assumption about whether any pair of hidden states s_i and s_j represents the same state or not. In certain cases, however, this information is available (e.g., traces drawn from the same state) and can be used. Action traces extended with such *state equalities* are called *extended traces*. It is useful to represent sets of extended traces T as graphs G_T. In these graphs, the nodes represent the hidden states, and two states *known* to represent the same state can be merged into a single node. The edges of the graph G_T are labeled with the actions mapping one state into the other. We will refer to both plain traces and extended traces as traces, and make their difference explicit when relevant.

SIFT: Learning from STRIPS Action Traces

The SIFT algorithm learns lifted STRIPS models, including the domain predicates, from STRIPS action traces, under the assumption that the hidden model is *well-formed*, in the sense that it does not add atoms that are already true, nor deletes atoms that are already false (Gösgens, Jansen, and Geffner 2025). The same assumption is made in SYNTH. The key idea in SIFT is that the domain predicates p in a hidden STRIPS domain $\mathcal{D}$ can be represented in a suitable manner as *features*, and they can be recovered from action traces T alone in time that is linear in $|T|$. For example, the atoms $\mathrm{at}(c)$ representing the position of the agent in a grid are affected only by the lifted actions *move*(x, x') when $x' = c$ or $x = c$. In the first case, $\mathrm{at}(c)$ becomes true; in the second case, it becomes false. The at predicate is represented indeed by the feature $f_{\mathrm{at}} = \langle 1, \{\mathit{move}[1], \mathit{move}[2]\} \rangle$ of arity 1 which takes one argument c, and which is affected only by actions *move*, when c is the first or second argument of *move*. Otherwise, the atom $\mathrm{at}(c)$ is not affected. More generally, a feature f is a pair $f = \langle k, B \rangle$ where k is the arity of the feature, and B is a set of action patterns $a[t]$ that affect f, where a is an action name in $\mathcal{D}$ and t is a tuple $t = [t_1, \ldots, t_k]$ of k indices t_i that range over the argument indices of the action a. An action pattern $a[t]$ in B says that the lifted atom $f(x_{t_1}, \ldots, x_{t_k})$ is an effect of the action $a(x_1, \ldots, x_{n_a})$.

A feature $f = \langle k, B \rangle$ represents a hypothetical domain predicate and the ways in which the actions affect it. The feature represents an actual domain predicate if it is *consistent* with the domain traces, and this consistency check reduces to a fast 2-CNF consistency test (Gösgens, Jansen, and Geffner 2025). The consistent features $f = \langle k, B \rangle$ are transformed into consistent *signed features* $f' = \langle k, A, D \rangle$ where the set of action patterns B is split into add patterns in A, and delete patterns in D, for $B = A \dot\cup D$. SIFT generates all possible action patterns $a[t]$ of arities $k = 1, \ldots, max$, $t = [t_1, \ldots, t_k]$, where *max* is the max arity of an action in the traces, and each of the possible features $f = \langle k, B \rangle$ is checked for consistency individually. The features consistent with the traces encode the action effects in a learned domain $\mathcal{D}_L$ whose preconditions can be inferred from the traces as well. For a sufficiently rich set of traces T, the authors show that the learned domain $\mathcal{D}_T$ and the hidden domain $\mathcal{D}$ are equivalent.

SYNTH: STRIPS+ State-Action Traces

SYNTH learns from state-action traces drawn from a hidden STRIPS+ domain $\mathcal{D}$, and for this, it assumes not only that the domain is well-formed and that the action preconditions $\Phi(x, y, z)$ of action $a(x)$ determine uniquely the value of the z-variables in any state s where a ground instance $a(o)$ applies (a requirement in STRIPS+), but also that these preconditions are easy to evaluate. For this, each (existential) y_i variable in y must occur only once in $\Phi(x, y, z)$, and the preconditions $\Phi(x, y, z)$ must be *stratified* in the following sense: they can be written as a sequence of $Q^i(x, y, z^i)$ expressions called *subqueries*, each including the atoms that include the variable z_i and no variable z_j, $j > i$, $z = \langle z_1, \ldots, z_n \rangle$. Moreover, not only must $Q^n(x, y, z^n)$ determine the value of the z variables, as demanded by STRIPS+; stratification demands that each subquery $Q^i(x, y, z^i)$ determines the value of each z^i variable (Jansen, Gösgens, and Geffner 2025).

Under these conditions and with a suitable rich set of state-action traces T, SYNTH learns a domain $\mathcal{D}_T$ that is equivalent to the hidden domain $\mathcal{D}$. The key task is learning the subqueries $Q^i(x, y, z^i)$, $i = 1, \ldots, n$, that set the values of the z-variables uniquely in each state where an instance $a(o)$ of an action $a(x)$ is applied. The full procedure is given in the appendix, as Algorithm 2. The task is solved greedily, with no loss of completeness, starting with $i = 1$, initializing each subquery $Q^{i+1}(x, y, z^{i+1})$ to $Q^i(x, y, z^i)$, and extending it incrementally with lifted atoms q_{i+1} that can involve x and y variables, and must involve variable z_{i+1} but no z_j variable for $j > i + 1$. After each extension, the satisfiability of $Q^{i+1}(x, y, z^{i+1})$ is checked in each state s of the traces where an action $a(o)$ is applied with $x = o$. If for each state s, the formula is satisfiable (*valid* in the code), q_{i+1} is retained in $Q^{i+1}(x, y, z^{i+1})$, else it is discarded. The step finishes when $Q^{i+1}(x, y, z^{i+1})$ determines the value of z^{i+1} uniquely in all such states (marked as *determined* in the code). $Q^0(x, y, z^0)$ is empty, and the whole process finishes with $z = \langle z_1, \ldots, z_n \rangle$, when no other determined variable z_{n+1} with a denotation different than the z_j variables found can be added to z. Provided with the z_i variables and the corresponding precondition queries $Q^i(x, y, z^i)$, the problem reduces to the well-known problem of learning action models from full STRIPS states and actions, as the objects that bind to the z_i variables can then be regarded as explicit action arguments.

SIFT and SYNTH Revisited

SIFT and SYNTH are presented as model learning algorithms: the first learns STRIPS models from STRIPS action traces; the second learns STRIPS+ models from STRIPS+ state-action traces. Yet another view of the core part of both algorithms is possible. Given relational states s and an action $a(x)$, SYNTH learns queries $q_i : Q^i(x, y, z^i)$ that bind the z_i variables to unique objects in the preconditions of $a(x)$. These queries can be thought as *referring expressions* with unique denotations. SIFT, on the other hand, "invents" predicates (consistent with the traces) over the explicit action arguments. A key observation is that it is direct to use SIFT to "invent" predicates over implicit action arguments as well. The only change that is needed is that in the action patterns $a[t]$ used to define the features, the indices t_i of t must be allowed to point to an explicit action argument x_i or to an implicit action argument z_l, whose value is captured by its query q_l in the precondition of action a.

Let us recall that SIFT does not learn the hidden domain $\mathcal{D}$ exactly, but with a sufficient rich set of traces, it learns models $\mathcal{D}'$ that are *equivalent* to $\mathcal{D}$ in the following sense:

Definition 1. *A domain $\mathcal{D}'$ is equivalent to a domain $\mathcal{D}$ if positive action traces in $\mathcal{D}$ are positive action traces in $\mathcal{D}'$, and negative action traces in $\mathcal{D}$ are negative action traces in $\mathcal{D}'$, where*

- *An action trace τ is* negative *if a_i is an action in τ with precondition p such that an action a_j before a_i deletes p, and no action between a_j and a_i adds p.*
- *An action trace τ is* positive *if it is not negative.*

Namely, $\mathcal{D}$ and $\mathcal{D}'$ are equivalent if they accept and reject the same action traces; meaning that a feasible plan in $\mathcal{D}$ must be a feasible plan in $\mathcal{D}'$, and vice versa. This is indeed the definition that is used in (Gösgens, Jansen, and Geffner 2025) to validate the learned models. The authors suggest an alternative definition that is equivalent to this one, that says that $\mathcal{D}$ and $\mathcal{D}'$ are equivalent if they can be extended into the so-called maximal domain descriptions $\mathcal{D}_{max}$ and $\mathcal{D}'_{max}$, compatible with the traces, such that $\mathcal{D}_{max}$ and $\mathcal{D}'_{max}$ are equal up to predicate renaming. Note that *static predicates*, namely, those which are not affected by the actions do not play any role in these definitions as they are not a true property of the domain but of the instances (Gösgens, Jansen, and Geffner 2025).

SIFT+ and SYNTH+: Pushing the Envelope

A simple extension and combination of the ideas in SIFT and SYNTH yields a new algorithm SYNTH+ which is more powerful than both. SYNTH+ learns from state-action traces assuming that a given subset of predicates $\mathcal{P}' \subseteq \mathcal{P}$ is observable. If $\mathcal{P}'$ is empty, SYNTH+ becomes an extension of SIFT, which we call SIFT+; if $\mathcal{P}' = \mathcal{P}$, it is equivalent to SYNTH.

Algorithm 1: SYNTH+ for learning STRIPS+ domains. It uses suitable extension of SIFT to invent new predicates from explicit and implicit action arguments, and SYNTH to generate new queries from them and observable predicates if any

Input: Labeled graph $G = G_T$ of traces T
Input: Obs. predicates $\mathcal{P}_\omega$ if any; else SYNTH+ =SIFT+
Output: Learned STRIPS+ domain $\mathcal{D}_T$

function SYNTH+(G,$\mathcal{P}_\omega$)
  $G' \leftarrow G$, $Q \leftarrow \emptyset$, $d \leftarrow 1$
  while G' updated **and** $d < d_{max}$ **do**
    $\mathcal{P}_n \leftarrow$ SIFT (G') for normal features
    $\mathcal{P}_m \leftarrow$ SIFT for mutex features$(\mathcal{P}_n, G')$.
    $\mathcal{P} \leftarrow \mathcal{P}_\omega \cup \mathcal{P}_n \cup \mathcal{P}_m$
    $Q \leftarrow Q \cup$ SYNTH $(G', \mathcal{P})$.
    $G' \leftarrow update(G', Q, \mathcal{P})$ with new z_i^a/Q^i's
    $d \leftarrow d + 1$
  $\mathcal{A} \leftarrow$ Effects/precs from learned features $\mathcal{P}_n$, $\mathcal{P}_m$
  $\mathcal{A} \leftarrow$ Add effects/precs over from obs. predicates $\mathcal{P}_\omega$
  return $\mathcal{D} = \langle \mathcal{P}, \mathcal{A} \rangle$

The interesting and novel cases are thus when $\mathcal{P}' \subset \mathcal{P}$. The pseudo-code for SYNTH+ is shown in Alg. 1 and will be explained below. For the integration of SIFT and SYNTH to be synergistic, however, a suitable extension of SIFT is needed: the ability to handle a new type of feature that we call *mutex features*, as they will represent predicates p of arity k in which the last, o_k, argument is determined by the previous ones $o_1, \ldots, o_{k-1}$. These features can be learned from action traces alone, very much as normal SIFT features, with two differences spelled out below.

Mutex Features

A mutex feature $f = \langle k, A, D \rangle$ is a signed feature of arity k where the action patterns $a[t]$ in A are positive, i.e., add f-atoms, and the action patterns $b[t']$ in D are negative, i.e., delete f-atoms. However, unlike the signed features that result from the consistency checks in SIFT where the arity of the action patterns in both A and D is k; in mutex features, the arity of the delete patterns $b[t']$ is $k-1$. The k-th argument is implicit, under the assumption, that has to be verified, that the k-th argument of an f-atom is *determined* by the other arguments. Intuitively, while a *consistent mutex feature* $f = \langle k, A, D \rangle$ expresses that the atom $f(x_{t_1}, \ldots, x_{t_k})$ is a positive effect of the action $a(x)$ with $x = \langle x_1, \ldots, x_n \rangle$ for a positive pattern $a[t]$ in A; the atom $f(x'_{t'_1}, \ldots, x'_{t'_{k-1}}, z_1)$ is a negative effect of the action $b(x')$ with $x' = \langle x'_1, \ldots, x'_m \rangle$ for the negative pattern $b[t']$ in D. This effect makes use of a z_1 variable whose value is determined by the STRIPS+ precondition $f(x_{t'_1}, \ldots, x_{t'_{k-1}}, z_1)$.

Definition 2 (Mutex Feature)**.** *A mutex feature $f = \langle k, A, D \rangle$ is a triplet given by a non-negative integer k, the arity of the feature, and sets of positive action patterns $a[t]$ in A of arity k, and negative patterns $b[t']$ in D of arity $k-1$.*

For example, the mutex feature $f = \langle 1, A, D \rangle$ with the positive action patterns of arity 1, $A = \{\textit{moveto}[1]\}$, and the negative action patterns of arity 0, $D = \{\textit{moveto}[]\}$, will capture the unary predicate $\mathrm{at}(x)$ which can only be true for a single constant c in a state. Indeed, the feature expresses that the action *moveto*(x) makes the atom $f(x)$ true and $f(z)$ false, provided the precondition $\mathrm{at}(z)$ where z is a determined variable. If one can learn such features, then the action traces do not have to spell out as many arguments, and moreover, the resulting z variables can be used to capture implicit action arguments in other actions like *pick*(o) or *drop*$()$, as we will see.

The key question is how the consistency of mutex features can be established given a set of extended traces. A slight extension of the ideas used in SIFT will provide the answer.

Consider the graph G_T associated with a set of extended traces T. The mutex feature $f = \langle k, A, D \rangle$ is consistent with the traces T if two conditions hold: 1) every ground atom $f(o)$, $o = \langle o_1, \ldots, o_k \rangle$, gets a single truth value in every node n in G_T following the action patterns in A and D, and 2) for every pair of atoms $f(o)$ and $f(o')$ true in a node n, $o' = \langle o'_1, \ldots, o'_k \rangle$ with $o_k \neq o'_k$, it must be the case that $o_i \neq o'_i$ for some $i < k$. The second condition guarantees that in every node the value of the argument at position k of f is determined by the precedent arguments.[2]

Definition 3 (Consistent Mutex Feature)**.** *A mutex feature $f = \langle k, A, D \rangle$ is consistent over a set of extended traces T if 1) there is a consistent assignment of truth values to the ground atoms $f(o)$ affected by ground actions in T over the whole graph G_T, and 2) for any two ground atoms $f(o)$ and $f(o')$ true in a node n in G_T with $o_k \neq o'_k$, it holds that $o_i \neq o'_i$ for some $i < k$.*

Testing if a mutex feature $f = \langle k, A, D \rangle$ is consistent with the traces T can thus be done efficiently in time that is polynomial in the length of the traces as in SIFT.

Fully Observable Predicates

SYNTH+ learns from *partially observable state-action* traces of the form $\omega_0, a_1, \ldots, a_n, \omega_{n+1}$ where a_i is a STRIPS+ action, and ω_i is a *partial observation* of the hidden state s_i in the trace. We consider a crisp form of partial state observability where a given subset $\mathcal{P}^\omega \subseteq \mathcal{P}$ of predicate symbols p in the domain $\mathcal{D} = \langle \mathcal{P}, \mathcal{A} \rangle$ is *fully observable*, meaning that ω_i is given by all the p-atoms that are true in the hidden state s_i for predicates $p \in \mathcal{P}^\omega$.

Definition 4 (Full Predicate Observability)**.** *If a predicate $p \in \mathcal{P}$ in a* STRIPS+ *domain $\mathcal{D} = \langle \mathcal{P}, \mathcal{A} \rangle$ is* fully observable*,*

[2] A ground action $a(o)$ in a node n of an extended trace makes a ground feature $f(o')$ true in the following node n', for the mutex feature $f = \langle k, A, D \rangle$, if there is an action pattern $a[t]$ in A such that $o' = t[o]$, where $t[o] = \langle o_{t_1}, \ldots, o_{t_k} \rangle$ for $t = [t_1, \ldots, t_k]$. From the well-formed assumption it follows that $f(o')$ must be false in the node n. Likewise, a ground action $b(o)$ in node n makes the ground feature $f(o')$ false in the following node n', if $f(o')$ is true in n, and there is an action pattern $b[t']$ in D such that $t'[o]$ captures the first $k-1$ arguments of o'. Finally, if an edge from n to n' in G_T is labeled with a ground action $c(o)$, such that there are no $c[t]$ patterns in neither A nor D, the truth value of the ground atoms $f(o)$ in n propagates to n' and vice versa.

then in a state s of an instance of the domain, truth of all the ground p-atoms in s is observed.

Full predicate observability is thus different from full state observability, which arises when all domain predicates are observable, and also from a notion of local predicate observability introduced below.

SYNTH+ and SIFT+

The SYNTH+ algorithm is shown as Algorithm 1. SYNTH+ takes two arguments: the labeled graph $G = G_T$ of the set of extended traces over the hidden STRIPS+ domain, and the subset of fully observable predicates $\mathcal{P}^\omega$. If $\mathcal{P}^\omega$ is empty, SYNTH+ learns solely from action traces, and behaves like an extension of SIFT, that we call SIFT+. The nodes in the graph represent hidden states, and the edge labels express the ground actions that map one state into the next one. During the procedure, other labels are added to the nodes and edges like the truth values of the learned and observed p-atoms, and the new implicit action arguments z_i^a and the queries Q_a^i that define their unique denotations for action a. In the inner loop, SYNTH+ calls SIFT iteratively to invent new predicates using action patterns defined over the explicit action arguments, and the implicit action arguments found so far (initially none), and SYNTH uses these predicates for defining new implicit action arguments. The mutex and plain predicates invented by SIFT correspond to the mutex and plain features that have been found consistent with the traces in the graph. The consistent features, however, yield more than predicates: they also encode the action effects, through the action patterns defining the features. The effects of the actions over the observed predicates in $\mathcal{P}^\omega$ are computed following the simpler procedure in SYNTH, because such effects can be computed over state transitions and not over full trajectories. Yet, SIFT eventually "rediscovers" such predicates, as they also correspond to features consistent with the traces. The difference is that observable predicates can be used from the beginning in query-preconditions to define implicit action arguments. The inner loop in SYNTH+ can reach a fixed point, but can also keep producing more implicit action arguments and more predicates. This process can be stopped in two ways: using negative traces in training or using a bound d_{max} on the number of iterations. If d_{max} is an upper bound on the length of the longest paths in the *dependency graph* of the the hidden domain (see below), the use of this bound does not affect the completeness of SYNTH+.

Properties

The scope of SIFT+ and SYNTH+, i.e., the class of STRIPS+ domains that they can learn correctly, is given by the dependency graph of the domain and the set of predicates that are observable. The dependency graphs have to be acyclic so that SIFT+ and SYNTH+ can learn the predicates one at a time.

Definition 5. *The directed* dependency graph $G^\omega_\mathcal{D} = (V, E)$ *associated with a* STRIPS+ *domain $\mathcal{D} = \langle \mathcal{P}, \mathcal{A} \rangle$ with observed predicates $\mathcal{P}^\omega$ and action schemas $\mathcal{A}$ is defined over vertices which represent the implicit variables z_i^a, $i = 0, \ldots, n_a$, for action schemas $a \in \mathcal{A}$, and non-observed predicates $p \notin \mathcal{P}^\omega$. The edges of this graph are:*

- ***Effects:*** *From p to z_i^a for $p \notin \mathcal{P}^\omega$ if there is a p-effect of action a that involves the variable z_i^a,*
- ***Preconditions:*** *From z_i^a to p if there is a p-atom in the subquery $Q^i(x, y, z^i)$ of action a,*
- ***Stratification:*** *From z_j^a to z_i^a if there is a precondition atom of a that involves the variables z_i^a and z_j^a, $j > i$.*

The difference between SYNTH+ and SIFT+ is that the former observes predicates while the latter does not observe any predicate, so for $\mathcal{P}^\omega = \emptyset$, SYNTH+ becomes SIFT+. This extra knowledge extends the conditions under which a domain is learned, and reduces the dependency graph $G^\omega_\mathcal{D}$. A STRIPS+ domain $\mathcal{D} = \langle \mathcal{P}, \mathcal{A} \rangle$ with observations $\mathcal{P}^\omega$ is *acyclic* if the graph $G^\omega_\mathcal{D}$ is acyclic; namely, it does not contain a directed path from a node to itself. If we let the *rank* of an acyclic domain represent the length of the longest path in the dependency graph, the condition under which a STRIPS+ domain can be learned from action traces alone can be expressed as:

Theorem 6. *If the hidden* STRIPS+ *domain $\mathcal{D} = \langle \mathcal{P}, \mathcal{A} \rangle$ is well-formed, strongly connected, and acyclic with rank bounded by d_{max}, then there is a set of extended traces from which the algorithm* SYNTH+ *will learn a domain equivalent to $\mathcal{D}$.*

A proof sketch of this theorem can be found in the appendix. In this theorem, a domain $\mathcal{D}$ is said to be strongly connected if in the instances of $\mathcal{D}$, every state s reachable from the initial state s_0 can reach s_0 again. The condition of acyclicity is needed so that the predicates can be learned one at a time, while the strong connectedness ensures that there are traces that can test the consistency of any feature. [3]

Note that if all predicates are observed, the *acyclicity* of $G^\omega_\mathcal{D}$ corresponds exactly to the conditions under which the STRIPS+ domain $\mathcal{D}$ is *stratified.* Indeed, if a predicate p in the hidden domain $\mathcal{D}$ is fully observable, it can be used in preconditions without having to be learned. Moreover, the effects on observable predicates can be determined later, without inducing dependencies in the graph $G^\omega_\mathcal{D}$. Detailed examples of both SIFT+ and SYNTH+ can be found in the appendix.

Local Predicate Observability

Full observability of a predicate p assumes that the learner has access to all the ground p-atoms true in a state. If the learner is the agent that is acting in the world, this is not a realistic assumption, and fortunately it is not necessary either. For example, an agent moving in a grid may observe the true atoms $\mathrm{at}(x)$ in the state, but only the atoms $\mathrm{adjacent}(x, x')$ involving the single value of x for which $\mathrm{at}(x)$ is true. We refer to this alternative form of observability as *local observability* and define it in a domain-independent way as follows:

[3] These are all sufficient conditions for the proof to hold, not necessary conditions. In the experiments below, the algorithm learns domains that are not strongly connected, and d_{max} can be set to infinity as, at some point, no new predicates or implicit arguments are found.

	Data							SIFT+ (top) / SYNTH+ (bottom)							Verification				SYNTH	
Domain	$\mathcal{P}^f$	$\mathcal{P}^s$	O	L	$\lvert x'\rvert$	$\mathcal{P}^f_\omega$	$\mathcal{P}^s_\omega$	$\lvert z\cap x'\rvert$	$\lvert z\setminus x'\rvert$	F^m	F^m_c	F	F_c	$\mathrm{T_L}$	$\mathrm{O_V}$	$\mathrm{S_V}$	$\mathrm{T_V}$	$\%\mathrm{V}$	$\lvert z\cap x'\rvert$	$\lvert z\setminus x'\rvert$
blocks3	3	1	6	$1k$	7	0	0	2	0	66	2	5	4	8 s	7	$1k$	101 s	100%	2	0
blocks4	5	0	7	$1k$	6	0	0	3	0	21	3	12	8	5 s	8	$1k$	68 s	100%	3	0
delivery	3	1	13	$7k$	9	0	0	4	0	101	5	7	5	22 s	14	$5k$	313 s	100%	4	0
driverlog	4	2	11	$61k$	19	0	0	9	2	1505800	16	49	23	111620 s	13	$41k$	20198 s	100%	9	0
ferry	4	1	10	$2k$	6	0	0	4	0	27	3	6	4	4 s	11	$2k$	64 s	100%	4	0
grid	6	6	13	$13k$	13	0	0	8	0	769	7	11	7	109 s	17	$9k$	1415 s	100%	9	4
gripper	4	4	11	$2k$	8	0	0	4	2	414	28	14	14	18 s	12	$2k$	203 s	100%	5	3
gripper4	4	4	13	$2k$	8	0	0	4	0	66	5	5	5	9 s	14	$2k$	113 s	100%	4	0
hanoi	2	1	6	$1k$	3	0	0	1	0	29	2	3	3	8 s	7	$1k$	140 s	100%	1	0
logistics	2	8	14	$23k$	13	0	0	5	0	561	9	7	7	98 s	19	$21k$	10028 s	100%	6	7
miconic	3	3	7	$1k$	8	0	0	4	2	102	9	8	8	2 s	11	$1k$	55 s	100%	6	0
n-puzzle	2	3	10	$6k$	16	0	0	8	0	5006	41	36	22	152 s	23	$10k$	6082 s	100%	16	0
c-puzzle	2	4	17	$1k$	12	0	0	4	0	791	3	16	2	39 s	31	$1k$	1153 s	100%	12	0
sokoban	2	2	15	$5k$	5	0	0	2	0	61	1	7	3	148 s	23	$5k$	3803 s	100%	3	0
sokoban	2	2	15	$6k$	5	0	2	3	0	61	1	7	3	453 s	24	$5k$	12331 s	100%	3	0
logistics*	2	8	14	$25k$	13	0	1	6	4	3833	16	11	11	1385 s	18	$21k$	56344 s	100%	6	4
s-delivery	4	2	19	$7k$	12	1	2	11	0	137	3	6	4	492 s	20	$5k$	12538 s	100%	11	0

Table 1: Results table. Table of results when learning a STRIPS+ domain $\mathcal{D}$ from *extended* traces of size L from an instance with O objects of a STRIPS domain $\mathcal{D}'$ containing $|x'|$ action arguments, $\mathcal{P}^f$ *non-static* and $\mathcal{P}^s$ *static* predicates. Hereby $\mathcal{P}^f_\omega$ denotes the number of fluent predicates and $\mathcal{P}^s_\omega$ the number of static predicates observable to SYNTH+, always zero for the SIFT+ cases. $|z\cap x'|$ denotes the number of STRIPS arguments not explicit in the trace but recovered by the algorithms, and $|z\setminus x'|$ the number of additional learned arguments not contained in $\mathcal{D}'$, with SIFT+/ SYNTH+ columns showing our results and SYNTH columns showing the corresponding numbers reported for SYNTH (Jansen, Gösgens, and Geffner 2025). T_L is the learning time. F plain features and F^m mutex features were generated and tested for consistency, with F_c and F^m_c consistent features, respectively. Each learned domain was validated using 24 positive and 24 negative traces each of length $\mathrm{S_V}$ sampled from instances containing $\#\mathrm{O_V}$ objects. $\mathrm{T_V}$ is the verification time, and $\%\mathrm{V}$ the success rate.

Definition 7 (Local objects and atoms)**.** *The set of* local objects *in a state s over a* STRIPS+ *domain $\mathcal{D} = \langle \mathcal{P}, \mathcal{A}\rangle$ refers to the set of objects appearing as explicit or implicit arguments of the ground actions* applicable *in s. The* local atoms *in s are the ground atoms that involve a local object.*

While a *fully-observable* predicate p, reveals the set of all true p-atoms in a state s, a *locally observable* predicate p reveals the set of all true p-atoms which are local in s:

Definition 8 (Local Observability)**.** *If a predicate $p \in \mathcal{P}$ in a* STRIPS+ *domain $\mathcal{D} = \langle \mathcal{P}, \mathcal{A}\rangle$ is* locally observable*, then in a state s of an instance of the domain, the truth of all the ground p-atoms which are local in s is known.*

If $p(c, d)$ is a ground atom in an instance of a domain $\mathcal{D}$ where p is locally observable, then the truth of $p(c, d)$ is assumed to be known in a true hidden state s iff it is a local atom in s. If p is a fully observed predicate, on the other hand, there is no distinction between local and non-local p-atoms. The interesting point is that SYNTH+ preserves completeness even when some of the fully observed predicates in $\mathcal{P}^\omega \subseteq \mathcal{P}$ become *locally observable*:

Definition 9. *A* key *predicate p in a* STRIPS+ *domain $\mathcal{D} = \langle \mathcal{P}, \mathcal{A}\rangle$ is a predicate that appears in an atom of a lifted action precondition and which involves no explicit action arguments x_i, and a single implicit action argument z_i.*

For example, in the *n-puzzle*, the action *Right* can have no explicit argument in a STRIPS+ encoding with preconditions $\mathrm{atB}(z_1)$, *right*(z_2, z_1), $\mathrm{atT}(z_3, z_2)$. In such a domain, the predicate atB is a key predicate as the precondition atom $\mathrm{atB}(z_1)$ involves no explicit argument and a single z-argument. The predicates *right* and atT are no key predicates as they involve a pair of z-variables.

We call the resulting algorithm LOCAL SYNTH+. LOCAL SYNTH+ differs from SYNTH+ in two minor ways. When testing the *validity* of a precondition and the *determination* of a query $Q^{i+1}(x, y, z^{i+1})$ within the TEST* procedure shown in the query-expansion procedure in the appendix in Alg. 2, special care is taken of atoms $p(o)$ for locally observable predicates p that are not local in a state. The truth of such atoms is unknown in s, yet they can be assumed to be *false* for testing validity of a precondition, and *true* for testing whether a query ensures that z^{i+1} binds to a unique object in s.

Theorem 10 (LOCAL SYNTH+)**.** *Let $\mathcal{D} = \langle \mathcal{P}, \mathcal{A}\rangle$ be a well-formed and strongly connected hidden* STRIPS+ *domain, and let $\mathcal{P}^\omega \subseteq \mathcal{P}$ be a subset of predicates such that the* key predicates *in $\mathcal{P}^\omega$ are fully observable, and the others are* locally observable*. Then, if the dependency graph $G^\omega_\mathcal{D}$ formed by assuming that all predicates $\mathcal{P}^\omega$ are fully observable, is acyclic and of rank bounded by d_{max}, the algorithm* LOCAL SYNTH+ *will learn a domain equivalent to $\mathcal{D}$.*

The theorem is a result of the definition of local observability and the change to the TEST* procedure, which en-

sures that non-observed atoms in the state do not affect the relevant queries or introduce additional invalid queries.

Experiments

We evaluate SIFT+ and SYNTH+ on a set of STRIPS domains under different observability assumptions about the actions and/or states in the traces.

Experimental Setup: For each STRIPS domain, a corresponding STRIPS+ domain was constructed by removing arguments (and predicates) such that the algorithms could infer all missing arguments. As input, six *extended* traces over the same instance of these domains were generated. The only exceptions are *n-puzzle*, *logistics* and *logistics**, for which traces of different instances were used. We compare our results with those reported for SYNTH (Jansen, Gösgens, and Geffner 2025), and for additional experiments not included in the paper, we used their publicly available implementation (Jansen and Gösgens 2025). All experiments were repeated 25 times and run on 10 CPU cores with a clock rate ranging from 2.1GHz to 2.9GHz, using up to 450GB of memory. All data and the implementation used are publicly available (Gösgens and Jansen 2026).

Domains: We evaluated SIFT+ on the same set of *well-formed* planning domains used in (Gösgens, Jansen, and Geffner 2025), where n-puzzle and c-puzzle refer to the sliding-tile puzzle with locations defined as coordinates and cells, respectively (Jansen, Gösgens, and Geffner 2025). We additionally included a *gripper* instance with four rooms and three grippers (*gripper4*). SYNTH+ was evaluated on three domains: *sokoban*, the *delivery* domain from Equation 3, and *logistics**, which is equivalent to *logistics* except that cities can have more than one airport.

Translation to STRIPS+: The STRIPS+ domains, from which action traces were generated to test SIFT+, were generated automatically from the STRIPS domains (Jansen, Gösgens, and Geffner 2025). For testing SYNTH+, some STRIPS+ action arguments were rendered unobservable, and some state predicates were chosen to be observable instead.

Data: For each STRIPS+ domain $\mathcal{D}$, six *extended* traces were sampled using breadth-first search, starting from a random reachable state of an instance. For all domains but *logistics*, *logistics**, and the *n-puzzle*, a single instance was used to sample the traces. For efficiency reasons, in these three domains, multiple smaller instances were used instead.

Verification: For verification, we generate both positive and negative extended traces from an instance of the STRIPS domain that is different from the one used to learn $\mathcal{D}$. Negative traces are traces in which the last action is not applicable because a precondition on a fluent predicate is false in the corresponding state. Positive traces are traces in which all actions are applicable. Verification fails if in the learned model, a negative trace is found to be positive, or a positive trace is found to be negative. A verification rate of 100% means that all traces are classified correctly.

Analysis of SIFT+: The results in the upper part of Table 1 show that SIFT+ is able to learn correct domains (100% verification) in all domains from action traces alone, even though these traces convey, on average, only 55% of the action arguments contained in the original STRIPS domains. This can be seen in the table where column $|x'|$ shows the number of action arguments in the STRIPS domain and $|z \cap x'|$ shows the number of arguments "recovered" by SIFT+. For example, in *ferry*, the traces recover 4 action arguments compared to 6 in the STRIPS domain, meaning that only 2 action arguments are explicit in the STRIPS+ domain. Similarly, in *grid*, 8 arguments are recovered even though only 5 are provided explicitly in the traces. In both cases, SIFT+ successfully learns the correct domain. In some domains, SYNTH recovers significantly more arguments than SIFT+, for example in *c-puzzle*, where SYNTH can learn from action traces with actions with no arguments, but uses state observability.

Analysis of SYNTH+: The bottom part of Table 1 shows that SYNTH+ learns a correct domain in every case, using the same number of explicit action arguments as SYNTH, i.e., those in the STRIPS+ encoding, but observing significantly fewer predicates.

In *sokoban*, only the static predicates defining the grid are observed, and no fluent predicate is observed. Similarly, in *logistics**, only the static predicate encoding the relations between cities and locations is observed. In *delivery*, as in (3) above, only the fluent predicate $at(x_1)$ is observed along with the static grid predicates, whereas SYNTH uses all six fluent predicates. In all these cases, neither SYNTH nor SIFT+ can learn the domains provided with the same inputs.

Conclusions

For action model learning to become a practical, lifted alternative to model-based reinforcement learning, it must be able to use natural traces conveying minimal information about actions and states, revealing certain predicates only and no more action arguments than those which are necessary for selecting the actions. Recent effective model learning algorithms like SIFT and SYNTH learn from pure STRIPS action traces, and from full state-action traces in STRIPS+ respectively, but can't deal with full or local observability of selected state predicates. We have shown, however, that a suitable extension of SIFT for learning mutex features, called SIFT+, in combination with SYNTH, can address such tasks effectively, while providing the theoretical conditions under which the algorithms are complete. For this, the query-synthesis algorithm of SYNTH yields queries or referring expressions, which SIFT uses to invent new predicates, in a loop that generates more queries and more predicates. While SIFT+ learns from action traces alone in STRIPS+, and SYNTH+ from partially observable state-action traces, the only difference between the two algorithms is that the latter starts with a non-empty set of observable predicates. Algorithms that initially appeared to be very different, SIFT and SYNTH, are thus combined and extended in SYNTH+. Future work includes effective methods for learning static predicates and for using partial observability of the successor states as well to further reduce the number of action arguments that need to be observed.

Acknowledgments

We thank Jonas Reiter and Jakob Gebler for insightful comments and helpful discussions. The research has been supported by the Alexander von Humboldt Foundation with funds from the German Federal Ministry for Education and Research. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 885107). This project was also funded by the German Federal Ministry of Education and Research (BMBF) and the Ministry of Culture and Science of the German State of North Rhine-Westphalia (MKW) under the Excellence Strategy of the Federal Government and the Länder.

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*.

Asai, M.; and Fukunaga, A. 2018. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *AAAI*.

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

Bachor, P.; and Behnke, G. 2024. Learning Planning Domains from Non-Redundant Fully-Observed Traces: Theoretical Foundations and Complexity Analysis. In *Proc. AAAI*, 20028–20035.

Balyo, T.; Suda, M.; Chrpa, L.; Šafránek, D.; Gocht, S.; Dvořák, F.; Barták, R.; and Youngblood, G. M. 2024. Planning domain model acquisition from state traces without action parameters. *arXiv preprint arXiv:2402.10726*.

Bonet, B.; and Geffner, H. 2020. Learning first-order symbolic representations for planning from the structure of the state space. In *Proc. ECAI*.

Brafman, R.; and Tennenholtz, M. 2003. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3: 213–231.

Burchi, M.; and Timofte, R. 2025. Learning Transformer-based World Models with Contrastive Predictive Coding. In *Proc. Int. Conf. on Learning Representations (ICLR)*.

Callanan, E.; De Venezia, R.; Armstrong, V.; Paredes, A.; Kang, J.; Chakraborti, T.; and Muise, C. 2022. MACQ: A Unified Library for Action Model Acquisition. In *Proc. ICAPS (Demonstrations)*.

Chevalier-Boisvert, M.; Bahdanau, D.; Lahlou, S.; Willems, L.; Saharia, C.; Nguyen, T. H.; and Bengio, Y. 2019. BabyAI: A Platform to Study the Sample Efficiency of Grounded Language Learning. In *ICLR*.

Cresswell, S.; and Gregory, P. 2011. Generalised Domain Model Acquisition from Action Traces. *Proc. ICAPS*, 42–49.

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

Diuk, C.; Cohen, A.; and Littman, M. L. 2008. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, 240–247.

Geffner, H.; and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Morgan & Claypool Publishers.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated planning and acting*. Cambridge U.P.

Gösgens, J.; Jansen, N.; and Geffner, H. 2025. Learning Lifted STRIPS Models from Action Traces Alone: A Simple, General, and Scalable Solution. In *Proc. ICAPS*.

Gregory, P.; and Cresswell, S. 2015. Domain model acquisition in the presence of static relations in the LOP system. In *Proc. ICAPS*, volume 25, 97–105.

Gösgens, J.; and Jansen, N. 2026. SYNTH+ implementation used for the experiments. https://doi.org/10.5281/zenodo.19857762.

Hafner, D.; Lillicrap, T.; Norouzi, M.; and Ba, J. 2021. Mastering atari with discrete world models. In *Proc. Int. Conf. on Learning Representations (ICLR)*.

Jansen, N.; Gösgens, J.; and Geffner, H. 2025. Learning Lifted Action Models From Traces of Incomplete Actions and States. In *Proc. KR*.

Jansen, N.; and Gösgens, J. 2025. SYNTH implementation used for the experiments. https://doi.org/10.5281/zenodo.16792702.

Lamanna, L.; Saetti, A.; Serafini, L.; Gerevini, A.; Traverso, P.; et al. 2021. Online Learning of Action Models for PDDL Planning. In *IJCAI*, 4112–4118.

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 *Proc. AAAI*, 20159–20167.

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

Micheli, V.; Alonso, E.; and Fleuret, F. 2023. Transformers are Sample-Efficient World Models. In *Int. Conf. on Learning Representations (ICLR)*.

Rodriguez, I. D.; Bonet, B.; Romero, J.; and Geffner, H. 2021. Learning First-Order Representations for Planning from Black Box States: New Results. In *Proc. KR*, 539–548.

Sutton, R. S.; and Barto, A. 2018. *Reinforcement learning: an introduction*. The MIT Press. 2nd edition.

Verma, P.; Marpally, S. R.; and Srivastava, S. 2021. Asking the right questions: Learning interpretable action models through query answering. In *Proc. AAAI*, 12024–12033.

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.

Zettlemoyer, L. S.; Pasula, H.; and Kaelbling, L. P. 2005. Learning planning rules in noisy stochastic worlds. In *AAAI*, 911–918.

Zhuo, H. H.; and Kambhampati, S. 2013. Action-model acquisition from noisy plan traces. In *Proc. IJCAI*.

Appendix

The following appendix provides detailed examples of SIFT+ and SYNTH+, pseudocode for the query expansion of SYNTH, and a proof sketch of Theorem 6.

Proof Sketch

Proof Sketch Theorem 6. Since the domain has an acyclic dependency graph, the queries can be learned one at a time. As the hidden domain is assumed to be strongly connected, the trace can be extended such that, for every state and atom, there is a path reaching the state that adds or deletes the atom, and consequently, all atoms that may satisfy a query are known. In iteration i, all queries of rank i are learned, since their arguments are observed and the dependent predicates must be consistent with the traces. After d_{max} iterations, every query is recovered. Because d_{max} is finite and each iteration considers only finitely many queries, a finite set of traces suffices to rule out all unsatisfiable queries, so the learned domain contains only queries and predicates consistent with the hidden domain.

□

Examples: SIFT+

We consider two domains which are fully learned without observing any predicate, whose dependency graphs are shown in Figure 1.

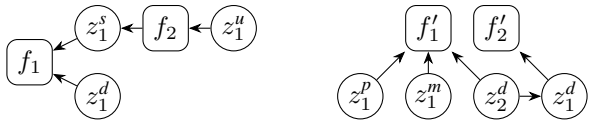
Blocksworld: The STRIPS actions $\underline{s}tack(x_1, x_2)$, $\underline{u}nstack(x_1, x_2)$, $\underline{p}ick(x_1)$, and $\underline{d}rop(x_1)$ are reduced automatically to STRIPS+ actions of lower arity: $stack(x_2^s)$, $unstack(x_1^u)$, $pick(x_1^p)$, $drop()$. All the action arguments dropped are recovered by SIFT+, without observing the states, using mutex features. The acyclic dependency between the features and z-variables can be seen in Figure 1a. First, the mutex feature

$$f_1 = \langle 1, \{unstack[1], pick[1]\}, \{stack[\,], drop[\,]\}\rangle,$$

is learned, which captures the predicate holding. Using preconditions $Q_1^p = f_1(z_1^d)$ and $Q_1^s = f_1(z_1^s)$, the block that is currently held can be inferred, resulting in actions $drop(z_1^d), stack(x_2^s, z_1^s)$. Only afterward, the mutex feature

$$f_2 = \langle 2, \{stack[2, 1]\}, \{unstack[1]\}\rangle$$

is learned, which captures the predicate on. Using $Q_1^u = f_2(x_1^u, z_1^u)$, the block below z_1^u is inferred, resulting in action $unstack(x_1^u, z_1^u)$. After this, no further arguments can be derived using referring expressions over mutex features.



(a) *Blocksworld* (b) *Delivery*

Figure 1: Dependency graphs for two domains

Delivery: Consider *delivery*, a domain with multiple agents and packages located on a grid of cells c. In STRIPS, this domain can be described using an action $\underline{m}ove(x_1^m, x_2^m, x_3^m)$, where agent x_1^m moves from x_2^m to x_3^m, and actions $\underline{p}ick(x_1^p, x_2^p, x_3^p)$, where agent x_1^p picks package x_2^p from cell x_3^p, and similarly $\underline{d}rop(x_1^d, x_2^d, x_3^d)$. In STRIPS+, the explicit actions can be reduced to $move(x_1^m, x_3^m)$, $pick(x_1^p, x_2^p)$, and $drop(x_2^d)$. All implicit arguments can be recovered using independent mutex features, as shown in the dependency graph in Figure 1b. Using these action schemas, the mutex feature

$$f_1' = \langle 2, \{move[1, 2]\}, \{move[1]\}\rangle$$

can be learned, capturing the STRIPS predicate $at(x_1, x_2)$. With the preconditions $Q_m^1 = f_1'(x_1^m, z_1^m)$ and $Q_p^1 = f_1'(x_1^p, z_1^p)$, additional arguments can be inferred, yielding the actions $move(x_1^m, x_3^m, z_1^m)$ and $pick(x_1^p, x_2^p, z_1^p)$, which now include the current position of the agent. Even without exploiting these newly inferred arguments, the mutex feature

$$f_2' = \langle 2, \{pick[2, 1]\}, \{drop[1]\}\rangle$$

can be learned, capturing the STRIPS predicate holding(p, a). Using the precondition $Q_1^d = f_2'(x_2^d, z_1^d)$, we obtain the action $drop(x_2^d, z_1^d)$, where the agent can be inferred from the package it is holding. Only after the agent that is holding the package is known, its position can be recovered using the precondition $Q_d^2 = f_1'(z_1^d, z_2^d)$, yielding the full action $drop(x_2^d, z_1^d, z_2^d)$. At this point, no further preconditions over mutex features yield new arguments, and the resulting action schemas are equivalent to the original STRIPS action schemas.

Examples: SYNTH+

The next examples illustrate that SYNTH+ strictly extends both SYNTH and SIFT+.

Delivery: Consider the delivery definition from Equation (3) with a single agent, where the agent's position ($at(x)$) and the adjacency relations ($leftof(x, y)/belowof(x, y)$) are observed, and only the action $pick(x_1)$ has an observed argument, namely the package that is picked. The agent's position can be inferred for each action a using the query $Q_a^1 = at(z_1)$, and for the move actions the next position can be inferred using the adjacency relations (e.g. $Q_{down}^2 = belowof(z_1, z_2)$ for action $down$). The mutex feature

$$f_1 = \langle 1, \{pick[1]\}, \{drop[]\}\rangle$$

infers the package currently held, and the query $Q_{drop}^2 = f_1(z_2)$ then infers the package being dropped. These queries recover all arguments in the hidden domain, so any fluent

Algorithm 2: Expands $Q^i(x, y, z^i)$ into $Q^{i+1}(x, y, z^{i+1})$ (Jansen, Gösgens and Geffner 2025)

Input: $Q(x, y, z^i) = \bigwedge_{j=1}^{i} Q_j(x, y, z^j)$ ▷ Valid query Q
Input: AS $= \{(a(o_i), s_i)\}_{i=1}^{n}$ ▷ State-action pairs
Output: $Q_{i+1}(x, y, z^{i+1})$ ▷ Valid extension of Q

function EXPAND($Q(x, y, z^i)$, AS)
  $Q \leftarrow \{p(w) \mid p \in P, w \in \{x, y, z^{i+1}\}, z_{i+1} \in w\}$
  $Q_0 \leftarrow \{q_d\}$ ▷ q_d is dummy query $true$
  while $Q_0 \neq \emptyset$ **do**
    $Q_{next} \leftarrow \{\}$
    for $q \in Q_0$ **do**
      for $q' \in (Q \setminus q)$ **do**
        $q'' \leftarrow q \wedge q'$
        res $\leftarrow$ TEST*$(Q(x, y, z^i), q'', AS)$
        if res $=$ *determined* **then**
          return q''
        else if res $=$ *valid* **then**
          $Q_{next} \leftarrow Q_{next} \cup q''$
    $Q_0 \leftarrow Q_{next}$
  return $Q^i(x, y, z^i)$ ▷ It can't be extended further

predicate SIFT could learn from full STRIPS actions can now be learned. Since all fluent predicates are learned and all static predicates are observed, SYNTH+ can infer every argument that SYNTH can infer from full states. By contrast, SYNTH cannot infer the package that is dropped since the predicate is missing and SIFT+ cannot infer the agent's position, so neither learns the correct domain.

Sokoban: Let the predicates $connected$, $connected_2$ be observed, as well as actions $up(x_1), push(x_1)$, where x_1 denotes the next position of the agent. The mutex feature

$$f_1 = \langle 1, \{move[1], push[1]\}, \{move[], push[]\}\rangle$$

infers the current position of the agent, and using the query $Q^1_{move} = Q^1_{push} = f_1(z_1)$ for both actions this position can be inferred. Afterward, the query $Q^2_{move} = connected(x_1, z_2), connected_2(z_1, z_2)$ infers the cell to which a box is pushed. Again all arguments are recovered and all predicates and queries can be learned. The examples are beyond the scope of both SIFT+ and SYNTH.