Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning
KR in Planning and Scheduling

# Learning Lifted Action Models From Traces of Incomplete Actions and States

**Niklas Jansen , Jonas Gösgens , Hector Geffner**
RWTH Aachen University
{niklas.jansen, jonas.goesgens, hector.geffner}@ml.rwth-aachen.de

**Abstract**

Consider the problem of learning a lifted STRIPS model of the sliding-tile puzzle from random state-action traces where the states represent the location of the tiles only, and the actions are the labels up, down, left, and right, with no arguments. Two challenges are involved in this problem. First, the states are not full STRIPS states, as some predicates are missing, like the atoms representing the position of the "blank". Second, the actions are not full STRIPS either, as they do not reveal all the objects involved in the actions effects and preconditions. Previous approaches have addressed different versions of this model learning problem, but most assume that actions in the traces are full STRIPS actions or that the domain predicates are all observable. The new setting considered in this work is more "realistic", as the atoms observed convey the state of the world but not full STRIPS states, and the actions reveal the arguments needed for selecting the action but not the ones needed for modeling it in STRIPS. For formulating and addressing the learning problem, we introduce a variant of STRIPS, which we call STRIPS+, where certain STRIPS action arguments can be left implicit in preconditions which can also involve a limited form of existential quantification. The learning problem becomes the problem of learning STRIPS+ models from STRIPS+ state-action traces. For this, the proposed learning algorithm, called SYNTH, constructs a stratified sequence (conjunction) of precondition expressions or "queries" for each action, that denote unique objects in the state and ground the implicit action arguments in STRIPS+. The correctness and completeness of SYNTH is established, and its scalability is tested on state-action traces obtained from STRIPS+ models derived from existing STRIPS domains.

## 1 Introduction

The problem of learning action models from data is fundamental in both planning and reinforcement learning. In classical planning, lifted models are learned from observed traces that may contain action and state information from a hidden STRIPS domain (Yang, Wu, and Jiang 2007; Arora et al. 2018; Aineto, Celorrio, and Onaindia 2019; Balyo et al. 2024), while in model-based reinforcement learning action models are learned from similar traces but without making assumptions about the structure of the hidden domain (Sutton and Barto 2018; Brafman and Tennenholtz 2003). As a result, model-based RL approaches have been successfully used in non-STRIPS domains like the Atari games (Micheli, Alonso, and Fleuret 2023; Hafner et al. 2021; Burchi and Timofte 2025) but the models learned are not lifted nor transparent.

In this paper, we aim to start exploring the middle ground between model-learning approaches in classical planning and reinforcement learning. A key step for this is to drop the assumption that the observed actions or states in the input traces come from a hidden STRIPS domain. This is because such types of traces make unrealistic assumptions about the information that the learning agent can actually perceive. Consider for example the domains NAVIG and SLIDING-TILE PUZZLE. In the first, an agent can move in a grid, one unit at a time; in the other, the "blank tile" can move in a grid in the same way. The observed actions in the two domains can be described uniquely in terms of the four labels UP, DOWN, LEFT, and RIGHT, yet these labels do not represent full STRIPS actions. The reason is that actions in STRIPS are forced to include as arguments *all the objects involved in atoms that change*. This means that in NAVIG, the actions must include the current and next location of the agent as arguments, while in the SLIDING-TILE PUZZLE, they must include the tile that is moved, along with its current and new location.

While these examples illustrate that it is not reasonable to assume that the observed traces contain full STRIPS actions, a similar argument can be made about full STRIPS states. Indeed, STRIPS encodings of the SLIDING-TILE PUZZLE need to include a predicate tracking the position of the "blank", but this predicate is not needed to represent the state of the world which is given by the positions of the tiles alone.

An alternative to traces with full STRIPS actions and/or full STRIPS states is to make both actions and states *partially observable* (Lamanna et al. 2025). Yet without futher restrictions, the learning problem is not well defined and may have no general solution. Our goals in this paper are more modest but they can be understood from this perspective as well. We consider a specific version of this general learning problem where certain STRIPS action arguments and predicates are fully observable, while the other action arguments and predicates are not observable at all. Yet from these partial observations we will be able to uncover the lifted STRIPS model in full.

For formulating and addressing the resulting model learning problem we introduce a variant of the STRIPS language, which we call STRIPS+, where certain STRIPS action argu-

ments can be left implicit in preconditions which can also involve a limited form of existential quantification. The learning problem becomes the problem of learning STRIPS+ models from STRIPS+ state-action traces, and the proposed learning algorithm will involve the construction of a sequence of precondition expressions or "queries" for each action, each one built from the observed predicates and denoting unique objects in the states where the action is executed. The correctness and completeness of the learning algorithm will be established, and its scalability will be tested on state-action traces obtained from STRIPS+ models obtained from existing STRIPS domains.

The rest of the paper is structured as follows. We review related work and the STRIPS language first, then introduce the new target language for learning, STRIPS+, and the model learning task and algorithm. Finally, we present the experimental results, conclusions, and challenges.

## 2 Related Work

**Learning from states and actions.** The problem of learning lifted STRIPS models has a long history in planning. In most cases, the input traces combine information about actions and states. While observability of the states can be partial or noisy, in almost all cases the observations reveal all the domain predicates and their arities (Yang, Wu, and Jiang 2007; Mourão et al. 2012; Zhuo and Kambhampati 2013; Arora et al. 2018; 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; Behnke and Bercher 2024). Likewise, the actions are normally full STRIPS actions with all the arguments, the exception being a recent SAT-based learning formulation where only the action names are observed (along with the states), with no information about either their arguments or their arity (Balyo et al. 2024). Interestingly, Lamanna et al. (2025) recently considered learning action models from state-action traces where both the states and the action arguments can be partially observable, yet the approach comes with no guarantees. Indeed, the formulation, which is based on observations about the atoms affected by an action, implies that action arguments that are just involved in preconditions and are not affected by the action, must be observable, at least sometimes; else such arguments cannot be recovered.[1]

**Learning from actions only.** Fewer works have considered the problem of learning lifted STRIPS models from traces containing actions only. The LOCM systems (Cresswell and Gregory 2011; Cresswell, McCluskey, and West 2013; Gregory and Cresswell 2015; Lindsay 2021) accept action traces as inputs, and outputs lifted domain descriptions, but their properties and scope are not clear. More recently, the SIFT algorithm, which uses the same input traces as LOCM, has been shown to be sound, complete, and scalable (Gösgens, Jansen, and Geffner 2024). The problem with SIFT is that the actions in the traces are assumed to be full STRIPS actions with all the action arguments spelled out. There is also a SAT approach to lifted model learning, that accepts state graphs, not traces, where the states are not observable and edges are labeled with action names and no arguments (Bonet and Geffner 2020; Rodriguez et al. 2021). This approach learns from very sparse information, but unlike SIFT and LOCM, does not scale up.

**Model-based reinforcement learning.** Model-based RL algorithms learn controllers by learning (stochastic) models first, without making further assumptions about the structure of the models (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. In particular, successful model-based RL approaches for the Atari games or Minecraft learn the action dynamics using transformers or recurrent 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 not lifted.

## 3 Background: STRIPS

A classical STRIPS planning problem is a pair $P = \langle D, I\rangle$ where $D$ is a first-order *domain* and $I$ contains information about the instance (Geffner and Bonet 2013; Ghallab, Nau, and Traverso 2016). The domain $D$ has a set of predicate symbols $p$ and a set of (lifted) action schemas $a(x)$ with preconditions, add, and delete effects $Pre(a(x))$, $Add(a(x))$, $Del(a(x))$ given by atoms $p(x_1, \ldots, x_k)$, where $p$ is domain predicate of arity $k$, and each $x_i$ is an argument of the action schema. The instance information is a tuple $I = \langle O, \mathit{Init}, G\rangle$ where $O$ is a set of object names or constants $c_i$, and *Init* and $G$ are sets of *ground atoms* $p(c_1, \ldots, c_k)$ denoting the initial and goal situations.

An action schema $a(x)$, where $x$ is a tuple of variables $x_1, \ldots, x_n$, is instantiated by consistently replacing the variables $x_i$ by constants $c_i$ in the instance. In the typed version of STRIPS, the variables and the constants have types, and variables are replaced by constant of the same type. The atoms $p(x)$ are called fluent if they appear in the effect of some action, and else they are called static. Static atoms appear in the initial situation and action preconditions, and affect just the action groundings.

A STRIPS problem $P = \langle D, I\rangle$ defines a state model $S(P) = \langle S, s_0, S_G, \mathit{Act}, A, f\rangle$ in compact form where the states $s \in S$ are *sets of ground atoms* from $P$ (assumed to

[1] As a concrete example, a correct model for an action like $move(p, t_1, t_2, l)$ which moves a package $p$ from truck $t_1$ to truck $t_2$ when both trucks are at location $l$ and package $p$ is in truck $t_1$, cannot be learned in this approach if the action argument $l$ is not observable, as the location $l$ cannot be identified from the effects of the action alone. In SYNTH, this is not a problem.

be the true atoms in $s$), $s_0$ is the initial state $I$, $S_G$ is the set of goal states $s$ such that $G \subseteq s$, *Act* is the set of ground actions in $P$, $A(s)$ is the set of ground actions whose preconditions are (true) in $s$, and $f(a, s)$, for $a \in A(s)$, represents the state $s'$ that follows action $a$ in the state $s$; namely $s' = ((s \setminus Del(a)) \cup Add(a))$.

By design, a STRIPS action can only affect the truth of an atom $p(x_1, \ldots, x_k)$ if the atom is an action effect, and hence if all of its arguments $x_i$ are action arguments. This implies for example, that if an action for picking up a block $x$ makes an atom $clear(y)$ true, then $y$ must be an action argument.

An action sequence $\tau = a_0, a_1, \ldots, a_n$ is applicable in $P$ if $a_i \in A(s_i)$ and $s_{i+1} = f(a_i, s_i)$, for $i = 0, \ldots, n$. The states $s_i$ are said to be reachable in $P$, and the action sequence $\tau$ is a plan for $P$ if $s_{n+1}$ is a goal state.

An action trace in $D$ is an applicable action sequence in some instance $P = \langle D, I \rangle$. Algorithms like LOCM and SIFT learn (lifted) STRIPS models from action traces alone. Others approaches learn from traces $s_0, a_0, s_1, \ldots$ that combine states and actions, with some information about actions or states missing, or corrupted by noise. In our setting, STRIPS models will be learned from state-action traces where some of the action arguments and some of the predicates in the state are not observable at all.

## 4 STRIPS+

We cast the problem of learning lifted STRIPS models from traces of incomplete STRIPS actions and states into the problem of learning models in a language that is more succint than STRIPS, that we call STRIPS+. STRIPS+ extends STRIPS by allowing *tuples of free variables* $y$ and $z$ in the action schemas $a(x)$ that are not among the explicit action arguments $x$. The variables in $y$ can only appear in action preconditions, while the variables in $z$ can appear in action preconditions and effects. The $z$ variables, however, have to be *determined* by the $x$ variables as spelled out below. This extension of STRIPS is not new and forms part of some of the PDDL standards (Haslum et al. 2019). In particular, the first PDDL standard (McDermott et al. 1998) supports the $z$ variables, which are declared via the keyword `:vars`.

**Definition 1.** *Action schemas $a(x)$ in* STRIPS+ *have (conjunctive) preconditions $Pre(a(x)) = \phi(x, y, z)$ with free variables among those of $x$, $y$, and $z$ which are pairwise disjoint sets of variables. The variables in $x$ and $z$ can appear in action effects. The value of the variables in $z$ must be* determined *by the values of the variables in $x$ as defined below.*

The variables in $x$ denote explicit action arguments of the action $a(x)$ in STRIPS+, and the variables in $z$ denote "implicit" action arguments captured by the preconditions. STRIPS actions are trivial STRIPS+ actions with an empty list of implicit action arguments; while STRIPS+ actions map into STRIPS actions with more arguments after pushing the $y$ and $z$ variables into explicit action arguments in STRIPS. For this translation and semantics to be valid, the value (denotation) of the $z$ variables that can appear in the action effects, must be *uniquely determined* by the value of the explicit action arguments $x$ and the action preconditions.

For example, in the SLIDING-TILE PUZZLE, the action $up(c_1, c_2, t)$ that moves tile $t$ from cell $c_2$ to cell $c_1$ can be modeled in STRIPS with these three arguments. In STRIPS+, on the other hand, the action $up$ can be modeled without *any* arguments, as the three explicit arguments in STRIPS, $c_1$, $c_2$, and $t$ can be recovered from the values of the three free $z$ variables $z = \{z_1, z_2, z_3\}$ in its precondition $\phi(x, y, z)$ given by the formula $blank(z_1) \wedge above(z_1, z_2) \wedge at(z_3, z_2)$. Indeed, this precondition is satisfied in each state $s$ where the action $up$ is applicable by a unique grounding of the $z_i$ variables, so that such variables can be regarded as *implicit arguments* of the $up$ action. This is all formalized below.

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 $o$. Formally:

**Definition 2.** *For a* STRIPS *or* STRIPS+ *domain $D$, let $\phi(x, y, z)$ refer to a conjunction of domain atoms with arguments from the three disjoint variable sets $x, y, z$ and let $s$ be the initial state of an instance $P = \langle D, I \rangle$ of $D$.*[2] *Then,*

- *A grounding of $\phi(x, y, z)$ in $P$ is an assigment $\sigma$ of variables in the formula to constants (objects) in the instance.*
- *A grounding* satisfies *the formula $\phi(x, y, z)$ in $s$ if the resulting ground atoms are all true in $s$.*
- *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 $x$ in a STRIPS+ action $a(x)$ with precondition $\phi(x, y, z)$ if the groundings that satisfy the formula must agree on the value (grounding) of $z$ when they agree on the value of $x$:

**Definition 3.** *The value of the variables in $z$ are* determined *by the values of the variables in $x$ in the precondition $\phi(x, y, z)$ of an* STRIPS+ *action $a(x)$ in a domain $D$, if in the initial state of any instance $P$ of $D$, there are no two satisfying groundings $\sigma$ and $\sigma'$ of $\phi(x, y, z)$ such that $\sigma(x) = \sigma'(x)$ and $\sigma(z) \neq \sigma'(z)$.*

In the precondition $\phi(x, y, z)$ of the action $up$ above given by the formula $blank(z_1) \wedge above(z_1, z_2) \wedge at(z_3, z_2)$, the three variables $z_i$ in $z = \{z_1, z_2, z_3\}$ are determined, as in any legal state $s$ over the sliding-puzzle domain $D$, there is a unique grounding for $z_1$ (blank position), for $z_2$ (cell above the blank), and for $z_3$ (tile at the cell which is above the blank).

In addition to assuming that the grounding of the $z$ variables in STRIPS+ action schemas $a(x)$ is uniquely determined by the grounding of the explicit action arguments $x$ and the preconditions as expressed in Definition 1, we assume, as in SIFT, that action effects are "well-formed" in the sense that they change the state; namely, the complement of the effects must be explicit or implicit action preconditions, so that no action adds an atom that is true, or deletes an atom that is false (Gösgens, Jansen, and Geffner 2024).

Determined variables express *state-invariants* of the domain; namely, that in the reachable states where a ground

[2] Notice that a reachable state of an instance $P$ is the initial state of another instance $P'$ that is otherwise like $P$.

action $a(o)$ of the lifted action $a(x)$ with precondition $\phi(x, y, z)$ applies, there is a unique grounding for $z$ among the (non-empty) groundings that satisfy $\phi(x, y, z)$ with $x = o$. In other words, the grounding of $z$ is a function $f_{a,s}(x)$ of $x$, the state $s$, and the action instance $a = a(o)$.

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 $x'$.

**Example:** NAVIG. The problem of navigating in an empty grid can be modeled in STRIPS via action schemas like $up(c, c')$, where $c$ and $c'$ are grid cell variables, preconditions are $at(c)$ and $above(c', c)$, and effects are $at(c')$ and $\neg at(c)$. In STRIPS+, the explicit action arguments $c$ and $c'$ can be made implicit through the use of the preconditions $at(z)$ and $above(z', z)$ where the variables $z$ and $z'$ are determined, as in every state $s$, there is a single atom $at(z)$ that is true, and a single (static) true atom $above(z', z)$ given $z$. The result is that the action $up$ can be modeled with no (explicit) arguments in STRIPS+.

**Example:** SLIDING-TILE PUZZLE. In STRIPS, the domain can be modeled by actions like $up(c, c', t)$ where $t$ is the tile that moves from cell $c'$ to cell $c$, and preconditions involving the atoms $at(t, c')$, $blank(c)$, and $above(c, c')$, where $blank$ tracks the position of the "blank". In STRIPS+, the three action arguments can be made implicit as shown above.

**Example:** BLOCKS. In STRIPS, the action $unstack(x_1, x_2)$ takes as arguments the blocks $x_1, x_2$, where $x_1$ is stacked on $x_2$. In STRIPS+, the action $unstack(x_1)$ can take instead a single explicit action argument $x_1$, as the variable $x_2$ can be captured by an implicit variable $z_1$ whose unique grounding is determined by the value of $x_1$ and the precondition $on(x_1, z_1)$ in any state where the action $unstack(x_1)$ is applied. The explicit argument $x_1$ cannot be rendered implicit because multiple blocks may be potentially unstacked. Yet the action $unstack(x_2)$, where $x_2$ denotes the block beneath the one to be picked would be a well-formed schema too, with the atom $on(z_1, x_2)$ in its precondition determining the unique grounding of the block $z_1$ to be unstacked.

# 5 The Learning Task

The learning task is to infer a lifted STRIPS+ domain from random state-action traces obtained from instances over a hidden STRIPS+ domain. We state the task formally below after introducing some restrictions on the class of hidden STRIPS+ models.

## 5.1 Target Fragment of STRIPS+

We cannot address the learning task in its full generality because determining whether a precondition formula $\phi(x, y, z)$ is satisfiable in a state is already NP-hard. We assume instead a class of hidden STRIPS+ domains $D$ whose precondition formulas $\phi(x, y, z)$ are *easy to evaluate*. For convenience, we will refer to formulas $\phi(x, y, z)$ as *conjunctive queries*. In these formulas the atom predicates are the domain predicates and the arguments are free variables from $x$, $y$, and $z$.

The restrictions below limit the expressive power of the target language, but every STRIPS problem is part of this STRIPS+ fragment, as every STRIPS problem $P = \langle D, I\rangle$ is a STRIPS+ problem with precondition $\phi(x, y, z)$, where the sets of variables $y$ and $z$ are empty.

The first restriction applies to the $y$ variables:

**Definition 4.** *A conjunctive query $Q(x, y, z)$ is* simple *if each $y_i$ variable appears only once in $Q(x, y, z)$.*

This means that the only constraints on $y$ variables are those occurring in single atoms, and moreover, no variable $y$ appears twice in such atoms either (this last condition could be relaxed though). The second restriction, *stratification*, is more interesting and applies to the $z$ variables:

**Definition 5.** *A conjunctive query $Q(x, y, z)$ is* stratified *if it can be expressed as the conjunction of conjunctive queries $Q_1(x, y, z^1)$, ..., $Q_n(x, y, z^n)$, each with one or more atoms, such that:*

- *Each variable $z_i$ appears in $z^i$ but not in $z^j$, $j < i$.*
- *If $Q(x, y, z)$ is satisfiable in a state $s$ with grounding $\sigma(x) = o$ and $\sigma(z_i) = c_i$, $i = 1, \ldots, n$, there is no grounding $\sigma'$ satisfying the prefix $Q_1(x, y, z^1)$, ..., $Q_i(x, y, z^i)$ with $\sigma'(x) = o$ and $\sigma'(z_j) \neq \sigma(z_j)$, for any $1 \leq j \leq i$, $i = 1, \ldots, n$,*

In other words, the query $Q(x, y, z)$ is *stratified*, not just if the $z$ variables are determined by the $x$ variables through the $Q$-formula, but if the value of each individual variable $z_{i+1}$ in $z$ is determined by the value of the variables $x \cup \{z_1, \ldots, z_i\}$ in the subformula $Q_{i+1}(x, y, z^{i+1})$. In fact, the variables $z$ being determined by $x$ means that if the query is satisfiable in a state $s$, there is a *unique value* $z = f_{a,s}(x)$ for the $z$ variables that satisfies the query for a given grounding of $x$, state $s$, and action instance $a = a(o)$. Computing these values, however, can be computationally hard. Stratification provides conditions under which this task is easy and can be solved *one variable $z_i$ at a time.*

A domain $D$ is said to be *stratified* if the action precondition formulas $\phi(x, y, z)$ are simple and stratified.

## 5.2 Task

The learning task is to infer a lifted STRIPS+ domain $D_L$ from random state-action traces obtained from instances $P = \langle D, I\rangle$ of a hidden STRIPS+ domain $D$. The learned domain $D_L$ does not have to be syntactically equivalent to the hidden domain $D$ but has to be semantically equivalent, meaning that the traces from $P = \langle D, I\rangle$ must be traces of $P_L = \langle D_L, I\rangle$ and vice versa.

**Definition 6** (Learning task)**.** *Given state-action traces of the form $s_0, a_0, s_1, a_2, \ldots, s_n, a_n$ from* STRIPS+ *instances $P = \langle D, I\rangle$ of a hidden* stratified *domain $D$, the task is to learn a domain $D_L$ such that the instances $P' = \langle D_L, I\rangle$ generate the same traces as $P$.*

Thus, the domain $D_L$ is to be learned from traces obtained from instances $P = \langle D, I\rangle$ of a stratified domain $D$, and $D_L$ and $D$ are deemed equivalent if instances $P_i = \langle D, I_i\rangle$ and $P'_i = \langle D_L, I_i\rangle$ generate the same traces.

# 6 The Learning Algorithm

We address the learning task in three parts: learning the preconditions $Q(x, y, z)$ of the actions $a(x)$ that bind the $z$ variables uniquely, learning the extra preconditions $Q'(x, y, z)$ that use these bindings but which do not constraint further their values, and learning the action effects.

## 6.1 Learning the Binding Preconditions $Q(x, y, z)$

The assumption that hidden action preconditions $\phi(x, y, z)$ are simple and stratified suggests a simple algorithm for learning a formula equivalent to $\phi(x, y, z)$ from the traces in two parts: a "conjunctive query" $Q(x, y, z)$ that binds the values of the $z$ variables, and a second part $Q'(x, y, z)$ that uses such bindings. The query $Q(x, y, z)$ is made up itself of conjunctive subqueries $Q_1(x, y, z^1)$, ..., $Q_n(x, y, z^n)$ such that:

1. **Stratification**: The formulas $Q_1(x, y, z^1)$, ..., $Q_n(x, y, z^n)$ stratify $Q(x, y, z)$ (Definition 5).
2. **Validity**: If an action $a(o)$ applies in a state $s$ in the traces, $Q(x, y, z)$ must be satisfiable with $x = o$ in $s$.
3. **Maximality**: $n$ is maximal; i.e., no other determined variables can be pushed into $z$, and no two variables $z^i$ can be merged into one (i.e., in some action application they denote different objects).

The maximality condition is needed so that the task of learning action preconditions and effects can be decoupled. Indeed, we learn first preconditions, and then, from them, the action effects. These effects may use some of the $z$ variables "found" in the first step, but not necessarily all of them. The query $Q(x, y, z)$ provides a *referring expression* for each of the variables $z_i$, which selects a unique value (grounding) for $z_i$ in any state $s$ where a ground action $a(o)$ applies. Due to the stratification, the implicit value for $z_i$ in $a(o)$ may depend on the values of the variables $z_j$ preceding $z_i$ in the ordering, but these values are also determined by the binding $x = o$ for $a(x)$ in $s$. In this process, expressions that refer to the same constant in all the states drawn from the same instance are discarded, as they do not truly denote functions of the state (for example, "the top-left corner cell in a grid").

The learning algorithm, that we call SYNTH, "synthesizes" the query $Q(x, y, z)$ one subquery $Q_i(x, y, z^i)$ at a time, with each subquery being constructed one lifted atom $q_{i,j}(x, y, z^i)$ at a time too, where a *lifted atom* is an atom whose arguments are free variables from $x$, $y$ and $z^i$. More precisely, SYNTH constructs sequences of atoms of the form

$$q_{1,1}(x, y, z^1), \ldots, q_{1,n_1}(x, y, z^1)\ ;$$
$$q_{2,1}(x, y, z^2), \ldots, q_{1,n_2}(x, y, z^2)\ ;$$
$$q_{m,1}(x, y, z^m), \ldots, q_{1,n_m}(x, y, z^m)$$

where $q_{i,j}$ denotes predicates observed in the traces, and $z^i$ contains the variable $z_i$, possibly the $z$ variables up to $z_i$, and no variable $z_j$, $j > i$. The $Q_i(x, y, z^i)$ expressions correspond to the conjunction of the lifted atoms $q_{i,*}(\cdot)$, and the conjunction of the expressions $Q(x, y, z^i) = Q_1(x, y, z^1)$, ..., $Q_i(x, y, z^i)$ must determine single values for the variables $z_j$ for $j \leq i$ and a given value for $x$.

**Algorithm 1** TEST: checks for unique grounding of $z_{i+1}$ in $Q_{i+1}(x, y, z^{i+1})$ over all $s$ where $a(o)$ applies, given unique grounding $\sigma$ for $z_1, \ldots, z_i$ in $Q(x, y, z^i)$

**Input:** $Q(x, y, z^i) = \bigwedge_{j=1}^{i} Q_j(x, y, z^j)$ ▷ Valid Query $Q$
**Input:** $Q_{i+1}(x, y, z^{i+1})$ ▷ Extension of $Q$
**Input:** AS $= \{(a(o), s)\}$ ▷ Relevant state-action pairs

**function** TEST($Q(x, y, z^i), Q_{i+1}(x, y, z^{i+1})$, AS)
  $Q(x, y, z^{i+1}) \leftarrow Q(x, y, z^i) \wedge Q_{i+1}(x, y, z^{i+1})$
  unique $\leftarrow$ true
  $V = \{x_1, ..., x_n, z_1, ..., z_i\}$ ▷ Variables in $x$, $z^i$
  **for** $(a(o), s) \in$ AS **do**
    ▷ *get $\sigma$ for $z^i$ from $Q(x, y, z^i)$ in $s$ and $x = o$*
    $\Sigma \leftarrow$ get_assignments($Q(x, y, z^{i+1}), a(o), s, \sigma$)
    **if** $|\Sigma| = 0$ **then return** *Not-Valid*
    **else if** $|\Sigma| \geq 2$ **then** unique $\leftarrow$ false
    ▷ *Else* $|\Sigma| = \{\sigma\} = 1$ *and unique keeps its value*
  **if** $\neg$ unique **then return** *Not-Determined*
  **else if** $\exists v \in V : \forall a(o), s \in AS : \sigma(z_{i+1}) = \sigma(v)$ **then return** *Subsumed*
  **else return** *Valid*

The generation of the ordered atom sequences $Q_i(x, y, z^i)$ that make up the query $Q(x, y, z)$ for satisfying Conditions 1–3 (stratification, validity, and maximality) is computed "greedily", as all the successful maximal queries end up denoting the same unique tuple of objects.[3] For this, only sequences of atoms are considered that preserve stratification and validity. The sequence becomes invalid when the resulting partial query $Q(x, y, z^{i+1})$ becomes unsatisfiable with $x = o$ in some state $s$ of the traces where the action $a(o)$ applies. In addition, atoms $q_{i+1,*}(x, y, z^{i+1})$ are considered in the sequence only after the preceding variable $z^i$ has been determined by the formula $Q_i(x, y, z^i)$ built so far (stratification). Finally, the computation finishes with the query $Q(x, y, z) = Q_1(x, y, z^1), \ldots, Q_n(x, y, z^n)$ when the atom sequence representing the query cannot be extended with an additional variable $z_{n+1}$ which is not equivalent to the explicit $x$ variables, or the implicit $z_i$ variables, $i \leq n$.

In short, starting with $i = 1$, SYNTH tries to append a new atom $q_{i,j}(x, y, z^i)$ to the sequence prefix (initially empty), while preserving validity; i.e., the resulting formula $Q(x, y, z^i)$ must remain satisfiable with $x = o$ when $a(o)$ applies in a state $s$. This process continues until a sequence of atoms is found for which the grounding of variable $z_i$ becomes unique for each action grounding $a(o)$ and state in which the action is applied. At that point, we have the subquery $Q_i(x, y, z^i)$ and all the ones preceding it.

The two main routines of SYNTH are displayed in Algorithms 1 and 2.[4] The first, TEST procedure, gets as input

[3] The algorithm can also be seen as a dynamic programming algorithm that builds the subqueries $Q_i(x, y, z^i)$ sequentially.

[4] The actual code is equivalent but more efficient.

**Algorithm 2** EXPAND: Extends $Q(x, y, z^i)$ with lifted atoms $p(w)$ to determine one more variable $z_{i+1}$

**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 $=$ *VALID* **then**
          **return** $q''$
        **else if** res $=$ *Not-Determined* **then**
          $Q_{next} \leftarrow Q_{next} \cup q''$
    $Q_0 \leftarrow Q_{next}$
  **return** *Maximal* ▷ $Q(x, y, z^i)$ can't be extended

a query $Q(x, y, z^i)$, a new subquery $Q_{i+1}(x, y, z^{i+1})$, and a set $AS$ of action-state pairs $(s, a(o))$ where an instance $a(o)$ of action $a$ applies in state $s$ in the given traces, and checks whether the query that conjoins the two makes the $z_{i+1}$ variable determined in all such states. For this, it computes the set of groundings $\Sigma$ of the variables in $z^{i+1}$ that satisfy this conjunction for $x = o$, assuming that there is a single grounding $\sigma$ for the variables $z_1, \ldots, z_i$ in $z^i$ that satisfies $Q(x, y, z^i)$ given $x = o$. If $|\Sigma| = 0$, it returns that the query extension is not valid, if $|\Sigma| > 1$, that it is not determined (no unique grounding), and if $|\Sigma| = 1$, it returns that it is valid, if the new variable $z_{i+1}$ does not have the same denotation as a variable in $x$ or $z^i$ over all state-action pairs $(a(o), s)$ in $AS$. Otherwise, the procedure returns that the query extension is subsumed.

Algorithm 2 displays the procedure EXPAND, which is the heart of SYNTH: it incrementally refines a precondition query $Q = Q(x, y, z^i)$, which initially contains only the *dummy query* $q_d$, that is always true and does not involve (constraint) any variables, by conjoining it with lifted atoms $q$, one at a time, until these atoms jointly form the new query component $Q_{i+1}(x, y, z^{i+1})$ that determines the unique grounding of the variable $z_{i+1}$. For this, the TEST procedure is called with $Q$, $q$, and the relevant set of state-action pairs $AS$ (instances $a(o)$ of $a$ in the traces and the states $s$ where are applied). The atom $q$ can be used to expand the query only if TEST returns "valid", and in this case the exapansion $Q_{i+1}(x, y, z^{i+1})$ is returned. If the algorithm TEST returns "not-determined", the algorithm EXPAND will expand $q$ in the next iteration. When the subquery is "non-valid" or "subsumed" we known that it cannot be used to expand $Q(x, y, z^i)$.

The algorithm SYNTH is complete in the following sense:

**Theorem 1.** *Let $\phi(x, y, z)$ be the precondition of the action $a(x)$ in the hidden, stratified domain, and let $z_i = c_i^{s,o}$ be the value of variable $z_i$ in the state $s$ of the traces where the action $a(o)$ applies, $i = 1, \ldots, n$. Then* SYNTH *returns a query $Q(x, y', z')$ from the traces with $z' = (z'_1, \ldots, z'_m)$ where each $z_i$ variable in $z$ is captured by a $z'_j$ variable in $z'$; namely, there is a function $\rho(z_i) = z'_j$ such that in all such states $s$, $Q(x, y', z')$ is satisfiable in $s$ with $x = o$, and in all the satisfying groundings $\sigma$, $\sigma(z'_j) = c_i^{s,o}$.*

The theorem doesn't imply soundness in the sense that groundings of $Q(x, y, z)$ that satisfy $x = o$ in the state $s$ of the traces, imply that the ground action $a(o)$ must be applicable in $s$. Indeed, for this, the precondition query $Q(x, y, z)$ needs to be extended with other atoms as shown below.

In the implementation of SYNTH, the lifted atoms $q_{i,j}(x, y, z^i)$ are ordered lexicographically, so that a single ordering of the atoms is considered in the construction of the subquery $Q_i(x, y, z_i)$. Moreover, these atoms are represented by atom patterns, following an idea from SIFT (Gösgens, Jansen, and Geffner 2024). Namely, a pattern like $p(\_, 2, i_z, 1, \_, 1_z)$ is used to represent the lifted atom $p(y_1, x_2, z_i, x_1, y_2, z_1)$ in $Q_i(x, y, z^i)$ where $i > 1$.

## 6.2 Learning the Extra Preconditions and Effects

The precondition formula $Pre_L(a(x))$ of the STRIPS+ action $a(x)$ in the learned domain $D_L$ conjoins the query $Q(x, y, z)$ obtained from the traces with a formula $Q'(x, y, z)$ that does not refine the binding of the $z$ variables, but which contains all lifted atoms that are true in all the states $s$ where an action $a(o)$ applies. This ensures that if $Pre_L(a(x))$ is satisfiable with $x = o$ in a trace, then the precondition $Pre(a(x))$ of the action $a(x)$ in the hidden domain $D$ will also be satisfiable with $x = o$, and hence that if the ground action $a(o)$ applies in the state $s$ of a trace in the learned domain $D_L$, it also applies in $s$ in the hidden domain. The reverse implication is also true, and if $a(o)$ applies in a state $s$ of a trace in the hidden domain $D$, it also is applicable in the same state $s$ in the learned domain $D_L$. Otherwise, the action $a(x)$ would feature a precondition in the learned domain that is not true in a state $s$ of the traces where an action $a(o)$ is done, and hence the precondition in $D_L$ would not be valid over the traces.

The formula $Q'(x, y, z)$ to augment the precondition query $Q(x, y, z)$ in $Pre_L(a(x))$ is obtained as the conjunction of the lifted atoms $p(x, y, z)$ whose arguments are variables $x_i$ from the lifted action $a(x)$, $z_i$ variables from $z$, and $y_i$ variables from $y$. The $y_i$ variables can only appear once in these lifted atoms and they are assumed to be existentially quantified. The $x_i$ and $z_i$ variables are free, and in a state $s$ where an action $a(x)$ is applied, their values are determined by the action arguments and the the precondition query $Q(x, y, z)$.

A lifted atom $p(x, y, z)$ is a *valid precondition* of $a(x)$ given the traces $T$, and hence pushed into $Q'(x, y, z)$ and $Pre_L(a(x))$, if the formula $\exists y'.p(x, y', z)$ is true in all the states $s$ of $T$ where an instance $a(o)$ of $a$ is applied, provided the binding $x = o$ for $x$ and the unique grounding for $z$ determined by the precondition query $Q(x, y, z)$.

Likewise, a lifted atom $p(x, z)$ is a *valid positive (resp.*

*negative) effect* of $a(x)$ given the traces $T$, and hence pushed into $Add(a(x))$ (resp. $Del(a(x))$), if the formula $p(x, z)$ is false (resp. true) in a state $s$ of $T$, where an action instance $a(o)$ of $a$ is applied, and true (resp. false) in the resulting state $s'$, provided the binding $x = o$ for $x$ and the unique grounding for $z$ determined by the precondition query $Q(x, y, z)$.

### 6.3 Properties

If $D_L$ denotes the learned model, namely, the action schemas $a(x)$ for the actions appearing in the traces $T$ with their learned preconditions and effects, it can be shown that $D_L$ is equivalent to the hidden domain $D$ provided that the set of traces $T$ is rich enough and that $D$ is stratified (i.e., the unique grounding of the $z$ variables can be determined one $z_i$ variable at a time):

**Theorem 2.** *For a suitable finite set of traces $T$ from the hidden (stratified) domain $D$, the learned domain $D_L$ is equivalent to $D$, meaning that the state-action traces resulting from any instance $P = \langle D, I\rangle$ of $D$, are traces of $P' = \langle D_L, I\rangle$, and vice versa.*

*Proof.* (Sketch) SYNTH learns queries $Q_i$ for each of the hidden $z_j$ variables in $D$ such that the hidden preconditions $\phi(x, y, z)$ and $Q_i$ pick up the same denotations. The resulting learned preconditions in $D_L$ may be different than in $D$, and the indices $i$ and $j$ may be different too, but the function $f_{a,s}(x)$ represented by both, that defines the unique grounding of the $z$ variables, would be the same. Also, by construction all the preconditions in $D$ are expressible with the $x, y, z$ variables, and hence will be captured in the learned domain $D_L$, which may include other preconditions too. Similar to preconditions, all effects are expressible with $x, z$ variables, and since the domain is assumed to be well-formed, all effects of the hidden domain are captured by the learned domain $D_L$ too. Finally, any invalid precondition will be rendered invalid through a single state in a trace, and there is a finite number of such invalid preconditions that can be constructed. □

### 6.4 Negation

The extension of SYNTH for learning *negated preconditions* is convenient and direct, and is implemented in the algorithm. The only change is that in negated lifted atoms $\neg p(x, y, z)$, the $y$ variables are interpreted as *universally* quantified; namely, as $\forall y.\neg p(x, y, z)$. As a result, a precondition $\neg p(x, y, z)$ of an action $a(x)$ is true in a state $s$ with $x = o$ and $z = o'$, if the formula $\forall y.\neg p(x, y, z)$ is true in $s$ with the bindings for the $x$ and $z$ variables provided by $o$ and $o'$.

For example, in the SLIDING-TILE PUZZLE, the position $z_1$ of the blank can be obtained without the predicate $blank(x_i)$ through the negated precondition $\neg at(y_1, z_1)$ which stands for the formula $\forall y_1.\neg at(y_1, z_1)$, as this formula is true only when $z_1$ represents the unique cell without a tile.

## 7 Experiments

We have tested SYNTH over a number of existing STRIPS domains $D'$. For this, we converted these domains into STRIPS+ domains $D$ by moving the explicit action arguments in $D'$ that are determined into implicit $z$ arguments in $D$. On average, as we will see, this reduces the number of (observed) action arguments during training by half. Then a single random state-action trace is sampled from a large instance $P = \langle D, I\rangle$ from $D$ to learn the domain $D_L$. The hidden and the learned domains are then compared over other test instances in a verification phase. We provide further details of the set up below, along with the results. A finer grained analysis of the results can be found in the next section. The experiments have been run on Intel(R) Xeon(R) Platinum 8160 CPUs running at 2.10GHz and the data and code are publicly available (Jansen and Gösgens 2025).

**Domains:** The domains are the ones considered in the SIFT paper (Gösgens, Jansen, and Geffner 2024): Blocks with 3 and 4 operators, Delivery, Driverlog, Grid, Ferry, Gripper, Hanoi, Logistics, Miconic, Sliding-tile puzzle, Sokoban. The Sliding-tile puzzle is in two versions: one with separate $x$ and $y$ coordinates denoted as $n$-puzzle, and the other with cells, denoted as $c$-puzzle. Sokoban-Pull is a variation of Sokoban, adding one action schema for a pull-action to make the resulting domain *dead-end free*. Dead-ends present a potential problem in the generation of data, as random traces are often trapped in parts of the state space.

**Translation into STRIPS+:** The traces are not generated from these STRIPS domains $D'$ but from their STRIPS+ translation $D$ where some of the explicit action arguments in STRIPS are pushed into implicit $z$ arguments. The algorithm for doing this translation automatically is a simplification of the query learning component of SYNTH, as the preconditions are given. In order to determine if an argument variable $x'_i$ in a STRIPS action $a'(x')$ from $D'$ is determined by other arguments and hence can become a $z$ variable in the encoding of the equivalent action $a(x)$ in $D$, we check if in all states of traces drawn from $D'$ where the STRIPS action $a'(o')$ is applied, the value of the argument $o'_i$ is unique given the preconditions. The preconditions and effects of the STRIPS action $a'(x')$ and the resulting STRIPS+ action $a(x)$ are the same except for the renaming of the variables $x'_i$ into $x$ and $z$ variables. There are no (existentially quantified) variables $y$ in $D$, but they can appear in the learned domains $D_L$.

**Data generation:** For each STRIPS domain $D'$, we pick an instance $P = \langle D, I\rangle$ from the STRIPS+ translation, and generate a long random trace with up to ten thousands of state-action pairs.[5] In some domains, small instances and short traces suffice to learn the domains correctly, in other cases, larger instance and/or longer traces are needed.

**Verification:** The correctness of the learned domains $D_L$ is assessed by sampling a number of reachable states $s$ and ground actions $a(o)$ in instances $P = \langle D, I\rangle$ where $D$ is the

[5] Traces from multiple instances could have been used too.

hidden domain. The applicability and effects of these actions are then compared in $D$ and $D_L$. A 100% verification rate indicate a full match. For testing also the translation from STRIPS to STRIPS+, the STRIPS actions of the original domain that are applicable in $s$ are also considered, and they must map to the same set of successor states. This comparison with the original STRIPS domain is not done however in the experiments.

**Results:** Table 1 shows the results of the experiments. The columns on the left show the domains, the number of objects in the instance used to generate the single trace per domain (#O), the length of the trace measured as the number of state-action pairs (#L), and the total number of action arguments in the original STRIPS domain $D'$ ($|x'|$), and the reduced number of action arguments in the STRIPS+ translation $D$ used to generate the traces ($|x|$). This is followed by the sum of this number $|x|$ and the total number of implicit action arguments $z_i$ learned along with their queries ($|x|+|z|$). The following columns show the total number of explicit but determined STRIPS arguments $x'$ that the learned $z$ variables fail to capture ($|x'/z|$), [6] and the total number of $z$ variables learned which do not capture any $x'$ variable ($|z/x'|$). This is extra information learned; namely, valid query expressions that define functions $f_{a,s}(x)$ over the traces that are just not used in $D_L$. This is followed by the total learning time (T), and the verification data: number of objects used in the verification instance (#$O_v$), number of action-state pairs tested (#SA), the verification time ($T_v$) and the score (%V). The data on the top part of the figure is about experiments where the full STRIPS+ states in $D$, which are equal to the full STRIPS states in $D'$, are used in the traces. The data on the bottom part is about the experiments where all atoms involving selected predicates were removed from the states in the traces (incomplete states).

**Analysis:** In all domains, SYNTH learns domains $D_L$ that verify 100%. The learning times run from a few seconds, to 1695 seconds in Driverlog. The times grow with the number of domain predicates, their arities, and the length of the traces. Both the length of the traces and the size of the instances used to generate the traces were selected so that SYNTH outputs the correct domains. This doesn't happen if the traces are too short or the instances are too small. The domains that required the longest traces are Driverlog and Grid. At the same time, the $n$-puzzle provides a good illustration of the size of the instances required for correct learning. The $c$-puzzle uses cells and the domain is learned correctly from traces of the 4x4 $c$-puzzle. The $n$-puzzle, on the other hand, uses separate $x$ and $y$ coordinates instead of cells, and requires traces from the larger 5x5 $n$-puzzle, as smaller instances resulting in invalid referring expressions ($z$ variables and their queries).

[6] In the translation from STRIPS $D'$ into STRIPS+ $D$, some $x'_i$ variables in the STRIPS actions $a'(x')$ are pushed into $z_j$ variables in the STRIPS+ actions $a(x)$ of $D$ (this is why $|x| < |x'|$ in general). These $z_j$ variables and their queries have to be learned in $D_L$, and then $z_j$ captures $x'_i$ if in all the states $s$ where $a(x)$ applies, $z_j$ and $x'_i$ represent the same object.

Table 1 shows that the actions in the traces contain on average half of the arguments of the original STRIPS actions (i.e., the column $|x'|$, which expresses the total number of STRIPS action arguments, is on average twice the value of the column $|x|$, which captures the total number of STRIPS+ action arguments used in the traces). More details about this below. The learned queries capture indeed all the "redundant" (determined) STRIPS action arguments (column $|x'\backslash z|$) and more (column $|z\backslash x'|$). For instance, in Gripper, there are just two rooms and two grippers, and $z$ variables are learned that pick "the other room" and "the other gripper", which are not used in the learned preconditions or effects of the actions.

The bottom part of the table shows the results when learning from *incomplete states* in four domains where all atoms involving selected predicates were removed from the traces. The $blank$ predicate is removed in the $c$-puzzle,[7] the *on-table* and *clear* predicates in Blocks, the *in-lift* predicate in Miconic, and the *on* predicate in Ferry. These predicates can be defined in terms of the other predicates and thus do not provide information about the state of the world, but about the atoms needed to get an STRIPS encoding. In STRIPS+, these predicates are learned as existentially quantified preconditions and are not needed in the traces.

## 8 Fine-grained Analysis

We provide an analysis of the learned queries that account for the missing action arguments in the traces, that become the implicit action arguments in the learned domains.

- **Gripper:** The STRIPS actions are $move(r, r')$, $grab(b, r, g)$ and $drop(b, r, g)$. The actions in the STRIPS+ traces are $move()$, $grab(b, g)$, and $drop(b)$. SYNTH learns the precondition queries that capture the omitted arguments which are involved in preconditions and effects. For example, the current and next rooms in $move$ are captured by the learned queries $Q_1 = \{at(z_1)\}$ and $Q_2 = \{\neg at(z_2)\}$. These queries also capture the rooms for the actions $pick$ and $drop$. The missing gripper of the action *drop* is obtained by the learned query $Q_3 = \{carry(b, z_3)\}$, and actually, the other gripper is also identified through the query $Q_4 = \{\neg carry(b, z_4)\}$. One can see that for $pick$ and $drop$, the referring expression for the "room in which the robot is not located" is derived, while for $drop$, the expression for the "gripper in which the ball is not held" is derived. These referring expressions and their corresponding $z$ variables, however, are not used in the learned preconditions or effects.
- **Sokoban:** The STRIPS actions are $move(c_1, c_2)$ and $push(c_1, c_2, c_3)$, which translate to the STRIPS+ actions $move(c_2)$ and $push(c_2)$. For both actions the location of the agent $c_1$ is obtained by $z_1$ with query $Q_1 = \{at(z_1)\}$, while $c_3$ is obtained by $z_2$ with query $Q_2 = \{adjacent_2(z_1, z_2), adjacent(c_2, z_2)\}$.

[7] Interestingly, in the $n$-puzzle, which is like the $c$-puzzle but with separate $x$ and $y$ coordinates, the same predicate cannot be removed, as the limited form of existential quantification in action preconditions in our STRIPS+ fragment is not expressive enough to recover it.

| | Data | | | | Learning | | | | | Verification | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Domain | #O | #L | $\|x'\|$ | $\|x\|$ | $\|x\|+\|z\|$ | $\|z\|$ | $\|x'\backslash z\|$ | $\|z\backslash x'\|$ | T | $\#O_V$ | $\#S_V$ | $T_V$ | %V |
| blocks3 | 5 | 250 | 7 | 5 | 7 | 2 | 0 | 0 | $0.82s$ | 6 | 1200 | $80.56s$ | 100% |
| blocks4 | 5 | 250 | 6 | 3 | 6 | 3 | 0 | 0 | $1.43s$ | 6 | 1600 | $105.04s$ | 100% |
| delivery | 16 | 1000 | 9 | 5 | 9 | 4 | 0 | 0 | $101.4s$ | 24 | 1200 | $246.69s$ | 100% |
| driverlog | 63 | 10000 | 19 | 10 | 19 | 9 | 0 | 0 | $1695.32s$ | 148 | 2400 | $3757.22s$ | 100% |
| ferry | 8 | 100 | 6 | 2 | 6 | 4 | 0 | 0 | $0.96s$ | 10 | 1200 | $68.78s$ | 100% |
| grid | 50 | 10000 | 13 | 4 | 17 | 13 | 0 | 4 | $584.62s$ | 48 | 2000 | $1264.47s$ | 100% |
| gripper | 10 | 500 | 8 | 3 | 11 | 8 | 0 | 3 | $2.52s$ | 12 | 1000 | $96.14s$ | 100% |
| hanoi | 8 | 200 | 3 | 2 | 3 | 1 | 0 | 0 | $1.5s$ | 10 | 400 | $41.58s$ | 100% |
| logistics | 35 | 1000 | 13 | 7 | 20 | 13 | 0 | 7 | $28.25s$ | 45 | 1600 | $377.5s$ | 100% |
| miconic | 9 | 600 | 8 | 2 | 8 | 6 | 0 | 0 | $2.78s$ | 12 | 1600 | $104.14s$ | 100% |
| $n$-puzzle | 34 | 1000 | 16 | 0 | 16 | 16 | 0 | 0 | $498.28s$ | 34 | 1600 | $1123.36s$ | 100% |
| $c$-puzzle | 49 | 500 | 12 | 0 | 12 | 12 | 0 | 0 | $147.19s$ | 49 | 1600 | $569.38s$ | 100% |
| sokoban | 30 | 1000 | 5 | 2 | 5 | 3 | 0 | 0 | $20.7s$ | 30 | 800 | $72.45s$ | 100% |
| sokopull | 25 | 600 | 8 | 3 | 8 | 5 | 0 | 0 | $10.67s$ | 30 | 1200 | $80.51s$ | 100% |
| blocks3$^-$ | 5 | 250 | 7 | 5 | 7 | 2 | 0 | 0 | $0.73s$ | 6 | 1200 | $56.8s$ | 100% |
| ferry$^-$ | 8 | 100 | 6 | 2 | 6 | 4 | 0 | 0 | $0.54s$ | 10 | 1200 | $63.71s$ | 100% |
| miconic$^-$ | 9 | 600 | 8 | 2 | 8 | 6 | 0 | 0 | $2.0s$ | 12 | 1600 | $89.34s$ | 100% |
| $c$-puzzle$^-$ | 49 | 500 | 12 | 0 | 12 | 12 | 0 | 0 | $87.67s$ | 49 | 1600 | $488.87s$ | 100% |

Table 1: Table of results when learning a STRIPS+ domain $D_L$ from a trace of of length #L from hidden domain $D$ derived from a STRIPS domain $D'$ with #O objects. $|x'|$ is the total number of action arguments in $D'$, $|x|$ is the total number of action argument in $D$, $|z|$ is the number of learned implicit action arguments in $D_L$ and $|x|+|z|$ is their sum. $|x'\backslash z|$ is the number of action arguments in $D'$ not captured in $D_L$, while $|z\backslash x'|$ is the number of implicit action arguments in $D_L$ that are not in $D$ (they are not incorrect, just not used). T is the total time to learn $D_L$, $T_V$ is the time to verify $D_L$, $\#S_V$ is the number of sampled state-action pairs in the verification. All numbers are averages over 10 runs as traces are random. %V is the success rate of the verification. The rows in the second block show the experiments using incomplete STRIPS states in the traces where atoms involved selected predicates are removed.

- **c-Puzzle:** The STRIPS actions in $c$-puzzle are $up(c_1, c_2, t)$, $down(c_1, c_2, t)$, $left(c_1, c_2, t)$, and $right(c_1, c_2, t)$. In STRIPS+, the resulting actions take no arguments. Focusing on the first action, the position of the "blank" $c_2$ is picked up by a $z_1$ variable with query $Q_1 = \{\neg at(y_1, z_1)\}$, while the next position of the "blank" $c_1$ is picked up by $z_2$ with query $Q_2 = \{above(z_1, z_2)\}$, and the tile $t$ that is moved is picked up by $z_3$ with query $Q_3 = \{at(z_3, z_2)\}$. The predicate $blank$ is not used in a query and also is not needed in a precondition, since for example $up$ is applicable, if there is a cell above the $blank$ which is the case iff $Q_2$ is satisfied. Therefore, the predicate $blank$ does not need to be contained in the states such that the domain can be learned.
- **Blocks World 3:** The actions in STRIPS are $stack(b_1, b_2)$, $unstack(b_1, b_2)$, and $move(b_1, b_2, b_3)$. In STRIPS+, these actions reduce to $stack(b_1, b_2)$, $unstack(b_1)$, and $move(b_1, b_3)$. The argument $b_2$ is picked by a variable $z_1$ with query $Q_1 = \{on(b_1, z_1)\}$. In addition, the atoms $on\_table(x)$, $clear(x)$ are not needed in the state traces, as they can be learned as the negated preconditions of the form $\neg on(x, y)$ and $\neg on(y, x)$ respectively, that stand for $\forall y.\neg on(x, y)$ and $\forall y.\neg on(y, x)$.

## 9 Conclusions

The problem of learning lifted STRIPS model from action traces alone has been recently solved by the SIFT algorithm, a follow up to LOCM. The limitation is that the actions in the traces must come from a hidden STRIPS domain and include all the arguments. This means for example that to represent the action of unstacking a block $x$ in a trace, the location of the block $x$ must be conveyed as an extra argument. In this work, we addressed a new variant of the model learning problem which is more realistic, and closer to the settings used in model-based reinforcement learning where actions reveal a minimal number of arguments. The problem is formulated and solved as the task of learning lifted STRIPS+ models from STRIPS+ state-action traces. The resulting algorithm has a broad and crisp scope, where it is sound, complete, and scalable, as illustrated through the experiments.

One question that arises from this work is whether the proposed methods can be used to learn, for example, the deterministic dynamics of Atari-like video games. In this setting, actions have no explicit arguments and states are represented by the colors of the cells in a grid (pixels). One difference to learning the dynamics of the sliding-tile puzzles from grids (atoms $at(t, c)$) is that the objects in Atari can take up many cells. This seems to call for representation languages that are richer than STRIPS+, able to accommodate definitions (axioms) and partial observability, as one does not observe objects directly but cell colors.

## Acknowledgments

The research has been supported by the Alexander von Humboldt Foundation with funds from the German Federal

Ministry for Education and Research, by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovations programme (Grant agreement No. 885107), and by the Excellence Strategy of the Federal Government and the NRW State.

## References

Aineto, D., and Scala, E. 2024. Action model learning with guarantees. *arXiv preprint arXiv:2404.09631*.

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

Arora, A.; Fiorino, H.; Pellier, D.; Métivier, M.; and Pesty, S. 2018. A review of learning planning action models. *The Knowledge Engineering Review* 33.

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*.

Behnke, G., and Bercher, P. 2024. Envisioning a domain learning track for the ipc. In *Proc. ICAPS Workshop on the International Planning Competition*.

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)*.

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. 2024. Learning lifted strips models from action traces alone: A simple, general, and scalable solution. *arXiv preprint arXiv:2411.14995*.

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.

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

Haslum, P.; Lipovetzky, N.; Magazzeni, D.; and Muise, C. 2019. *An Introduction to the Planning Domain Definition Language*. Morgan & Claypool.

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.

Lindsay, A. 2021. Reuniting the locm family: An alternative method for identifying static relationships. In *Proc. ICAPS 2021 KEPS Workshop*.

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)*.

Mourão, K.; Zettlemoyer, L.; Petrick, R. P.; and Steedman, M. 2012. Learning strips operators from noisy and incomplete observations. In *Proc. UAI*, 614–623.

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.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence* 171(2-3):107–143.

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*.