# One-shot learning: From domain knowledge to action models

**Diego Aineto**[1] , **Sergio Jiménez**[1] , **Eva Onaindia**[1]

[1]Departamento de Sistemas Informáticos y Computación. Universitat Politècnica de València. Valencia, Spain

{dieaigar,serjice,onaindia}@dsic.upv.es

## Abstract

Most approaches to learning action planning models heavily rely on a significantly large volume of training samples or plan observations. In this paper, we adopt a different approach based on deductive learning from domain-specific knowledge, specifically from logic formulae that specify constraints about the possible states of a given domain. The minimal input observability required by our approach is a single example composed of a full initial state and a partial goal state. We will show that exploiting specific domain knowledge enable to constrain the space of possible action models as well as to complete partial observations, both of which turn out helpful to learn good-quality action models.

## 1 Introduction

The learning of action models in planning has been typically addressed with inductive learning data-intensive approaches. From the pioneer learning system ARMS [Yang *et al.*, 2007] to more recent ones [Mourão *et al.*, 2012; Zhuo and Kambhampati, 2013; Kucera and Barták, 2018], all of them require thousands of plan observations or training samples, i.e., sequences of actions as evidence of the execution of an observed agent, to obtain and validate an action model. These approaches return the statistically significant model that best explains the plan observations by minimizing some error metric. A model explains an observation if a plan containing the observed actions can be generated with the model and the states induced by this plan also include the possibly partially observed states. The limitation of posing model validation as an optimization task over a testing set of observations is that it neither guarantees completeness (the model may not explain all the observations) nor correctness (the states induced by the execution of the plan generated with the model may contain contradictory information).

Differently, other approaches rely on symbolic-via learning. The Simultaneous Learning and Filtering (SLAF) approach [Amir and Chang, 2008] exploits logical inference and builds a complete explanation through a CNF formula that represents the initial belief state, and a plan observation. The formula is updated with every action and state of the observation, thus representing all possible transition relations consistent with it. SLAF extracts all satisfying models of the learned formula with a SAT solver although the algorithm cannot effectively learn the preconditions of actions. A more recent approach addresses the learning of action models from plan observations as a planning task which searches the space of all possible action models [Aineto *et al.*, 2018]. A plan here is conceived as a series of steps that determine the preconditions and effects of the action models plus other steps that validate the formed actions in the observations. The advantage of this approach is that it only requires input samples of about a total of 50 actions.

This paper studies the impact of using mixed input data, i.e, automatically-collected plan observations and human-encoded domain-specific knowledge, in the learning of action models. Particularly, we aim to stress the extreme case of having a single observation sample and answer the question to whether the lack of training samples can be overcome with the supply of domain knowledge. The question is motivated by (a) the assumption that obtaining enough training observations is often difficult and costly, if not impossible in some domains [Zhuo, 2015]; (b) the fact that although the physics of the real-world domain being modeled are unknown, the user may know certain pieces of knowledge about the domain; and (c) the desire for correct action models that are usable beyond their applicability to a set of testing observations. To this end, we opted for checking our hypothesis in the framework proposed in [Aineto *et al.*, 2018] since this planning-based satisfiability approach allows us to configure additional constraints in the compilation scheme, it is able to work under a minimal set of observations and uses an off-the-shelf planner[1]. Ultimately, we aim to compare the informational power of domain observations (information quantity) with the representational power of domain-specific knowledge (information quality). Complementarily, we restrict our attention to solely observations over fluents as in many applications the actual actions of an agent may not be observable [Sohrabi *et al.*, 2016].

Next section summarizes basic planning concepts and outlines the baseline learning approach [Aineto *et al.*, 2018].

[1]We thank authors for providing us with the source files of their learning system.

Then we formalize our one-shot learning task with domain knowledge and subsequently we explain the task-solving process. Section 5 presents the experimental evaluation and last section concludes.

## 2 Background

This section formalizes the *planning model* we follow in this work and introduces the classical planning compilation for the learning of STRIPS action models [Aineto *et al.*, 2018]. Finally, the section formalizes *state-invariants*.

### 2.1 Classical planning with conditional effects

Let $F$ be the set of propositional state variables (*fluents*) describing a state. A *literal* $l$ is a valuation of a fluent $f \in F$; i.e. either $l = f$ or $l = \neg f$. A set of literals $L$ represents a partial assignment of values to fluents (without loss of generality, we will assume that $L$ does not contain conflicting values). Given $L$, let $\neg L = \{\neg l : l \in L\}$ be its complement. We use $\mathcal{L}(F)$ to denote the set of all literal sets on $F$; i.e. all partial assignments of values to fluents. A *state* $s$ is a full assignment of values to fluents; $|s| = |F|$.

A *classical planning action* $a \in A$ has: a precondition $\mathsf{pre}(a) \in \mathcal{L}(F)$, a set of effects $\mathsf{eff}(a) \in \mathcal{L}(F)$, and a positive action cost $cost(a)$. The semantics of actions $a \in A$ is specified with two functions: $\rho(s, a)$ denotes whether action $a$ is *applicable* in a state $s$ and $\theta(s, a)$ denotes the *successor state* that results of applying action $a$ in a state $s$. Then, $\rho(s, a)$ holds iff $\mathsf{pre}(a) \subseteq s$, i.e. if its precondition holds in $s$. The result of executing an applicable action $a \in A$ in a state $s$ is a new state $\theta(s, a) = (s \setminus \neg\mathsf{eff}(a)) \cup \mathsf{eff}(a)$. Subtracting the complement of $\mathsf{eff}(a)$ from $s$ ensures that $\theta(s, a)$ remains a well-defined state. The subset of action effects that assign a positive value to a state fluent is called *positive effects* and denoted by $\mathsf{eff}^+(a) \in \mathsf{eff}(a)$ while $\mathsf{eff}^-(a) \in \mathsf{eff}(a)$ denotes the *negative effects* of an action $a \in A$.

A *classical planning problem* is a tuple $P = \langle F, A, I, G\rangle$, where $I$ is the initial state and $G \in \mathcal{L}(F)$ is the set of goal conditions over the state variables. A *plan* $\pi$ is an action sequence $\pi = \langle a_1, \ldots, a_n\rangle$, with $|\pi| = n$ denoting its *plan length* and $cost(\pi) = \sum_{a\in\pi} cost(a)$ its *plan cost*. The execution of $\pi$ on the initial state of $P$ induces a *trajectory* $\tau(\pi, P) = \langle s_0, a_1, s_1, \ldots, a_n, s_n\rangle$ such that $s_0 = I$ and, for each $1 \leq i \leq n$, it holds $\rho(s_{i-1}, a_i)$ and $s_i = \theta(s_{i-1}, a_i)$. A plan $\pi$ solves $P$ iff the induced *trajectory* $\tau(\pi, P)$ reaches a final state $G \subseteq s_n$, where all goal conditions are met. A solution plan is *optimal* iff its cost is minimal.

We also define *actions with conditional effects* because they are useful to compactly formulate our approach for *goal recognition with unknown domain models*. An action $a_c \in A$ with conditional effects is a set of preconditions $\mathsf{pre}(a_c) \in \mathcal{L}(F)$ and a set of *conditional effects* $\mathsf{cond}(a_c)$. Each conditional effect $C \rhd E \in \mathsf{cond}(a_c)$ is composed of two sets of literals: $C \in \mathcal{L}(F)$, the *condition*, and $E \in \mathcal{L}(F)$, the *effect*. An action $a_c$ is applicable in a state $s$ if $\rho(s, a_c)$ is true, and the result of applying action $a_c$ in state $s$ is $\theta(s, a_c) = \{s \setminus \neg\mathsf{eff}_c(s, a) \cup \mathsf{eff}_c(s, a)\}$ where $\mathsf{eff}_c(s, a)$ are the *triggered effects* resulting from the action application (conditional effects whose conditions hold in $s$):

$$\mathsf{eff}_c(s, a) = \bigcup_{C \rhd E \in \mathsf{cond}(a_c), C \subseteq s} E,$$

### 2.2 Learning action models with classical planning

The *classical planning compilation* for the learning of STRIPS action models [Aineto *et al.*, 2018] receives as input an *empty* model (which contains just the *name* and *parameters* of each action schema), and a set of observations of plan executions. The compilation completes the *empty* model specifying the preconditions and effects of each action schema such that the validation of the completed model over the input observations is successful; i.e., there exists a plan computable with the completed model s.t. $\rho(s^o_{i-1}, a_i)$ and $s^o_i = \theta(s^o_{i-1}, a_i)$ holds for every observed state.

A solution plan to the classical planning problem that results from the compilation is then a sequence of:

- *Insert actions*, that insert preconditions and effects on an action schema.
- *Apply actions* that validate the application of the completed model in the input observations.

Figure 1 shows a solution to a classical planning problem resulting from the Aineto *et al.* 2018 compilation corresponding to the *blocksworld* [Slaney and Thiébaux, 2001]. In the initial state of that problem the robot hand is empty and three blocks (namely `blockA`, `blockB` and `blockC`) are on top of the table and clear. The problem goal is having the three-block tower `blockA` on top of `blockB` and `blockB` on top of `blockC`. The plan shows the *insert* actions for the `stack` scheme (steps $00-01$ insert the preconditions, steps $05-10$ insert the effects), the plan steps $02-04$ that insert the preconditions of the `pickup` scheme and steps $10-13$ that insert the effects of this scheme. Finally, steps $14-17$ is a plan postfix with actions that apply the programmed model to achieve the goals starting from the given initial state.

```
00:(insert_pre_stack_holding_v1)
01:(insert_pre_stack_clear_v2)
02:(insert_pre_pickup_handempty)
03:(insert_pre_pickup_clear_v1)
04:(insert_pre_pickup_ontable_v1)
05:(insert_eff_stack_clear_v1)
06:(insert_eff_stack_clear_v2)
07:(insert_eff_stack_handempty)
08:(insert_eff_stack_holding_v1)
09:(insert_eff_stack_on_v1_v2)
10:(insert_eff_pickup_clear_v1)
11:(insert_eff_pickup_ontable_v1)
12:(insert_eff_pickup_handempty)
13:(insert_eff_pickup_holding_v1)
14:(apply_pickup blockB)
15:(apply_stack blockB blockC)
16:(apply_pickup blockA)
17:(apply_stack blockA blockB)
18:(validate_1)
```

Figure 1: Example of a solution to a problem output by the classical planning compilation for the learning STRIPS action models.

## 3 *One-shot* learning of planning action models from *domain-specific knowledge*

We define the *one-shot* learning of planning action models from *domain-specific knowledge* as a tuple $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi\rangle$, where:

- $\mathcal{M}$ is the *initial empty model* that contains only the name and parameters of each planning action to be learned.

- $\mathcal{O}$ is a single learning example that represents the observation of a sequence of states generated with the aimed planning action model.
- $\Phi$ is a set of logic formulae defining *domain-specific knowledge* that constraint the set of possible states.

A *solution* to a learning task $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$ is a model $\mathcal{M}'$ s.t. there exists a plan computable with $\mathcal{M}'$ that is consistent with the the *initial empty model* $\mathcal{M}$, the single learning example $\mathcal{O}$ and the given *domain-specific knowledge* in $\Phi$.

### 3.1 The space of STRIPS action models

*A* STRIPS *action schema* $\xi$ is defined by: A list of *parameters* $pars(\xi)$, and three sets of predicates (namely $pre(\xi)$, $del(\xi)$ and $add(\xi)$) that shape the kind of fluents that can appear in the *preconditions*, *negative effects* and *positive effects* of the actions induced from that schema. Let be $\Psi$ the set of *predicates* that shape the propositional state variables $F$, and a list of *parameters*, $pars(\xi)$. The set of elements that can appear in $pre(\xi)$, $del(\xi)$ and $add(\xi)$ of the STRIPS action schema $\xi$ is the set of FOL interpretations of $\Psi$ over the parameters $pars(\xi)$ and is denoted as $\mathcal{I}_{\Psi,\xi}$.

For instance in a four-operator *blocksworld* [Slaney and Thiébaux, 2001], the $\mathcal{I}_{\Psi,\xi}$ set contains only five elements for the `pickup`($v_1$) schemata, $\mathcal{I}_{\Psi,pickup}$={`handempty`, `holding`($v_1$), `clear`($v_1$), `ontable`($v_1$), `on`($v_1$,$v_1$)} while it contains eleven elements for the `stack`($v_1$,$v_2$) schemata, $\mathcal{I}_{\Psi,stack}$={`handempty`, `holding`($v_1$), `holding`($v_2$), `clear`($v_1$), `clear`($v_2$), `ontable`($v_1$), `ontable`($v_2$), `on`($v_1$,$v_1$), `on`($v_1$,$v_2$), `on`($v_2$,$v_1$), `on`($v_2$,$v_2$)}.

Despite any element of $\mathcal{I}_{\Psi,\xi}$ can *a priori* appear in the $pre(\xi)$, $del(\xi)$ and $add(\xi)$ of schema $\xi$, in practice the actual space of possible STRIPS schemata is bounded by:

1. **Syntactic constraints**. STRIPS constraints require $del(\xi) \subseteq pre(\xi)$, $del(\xi) \cap add(\xi) = \emptyset$ and $pre(\xi) \cap add(\xi) = \emptyset$. Considering exclusively these syntactic constraints, the size of the space of possible STRIPS schemata is given by $2^{2\times|\mathcal{I}_{\Psi,\xi}|}$. *Typing constraints* are also of this kind [McDermott *et al.*, 1998].
2. **Observation constraints**. The *learning examples*, that in our case is the single observation of a sequence of states, depict *semantic knowledge* that constraints further the space of possible action schemata.

In this work we introduce a novel propositional encoding of the *preconditions*, *negative*, and *positive* effects of a STRIPS action schema $\xi$ that uses only fluents of two kinds `pre_e_`$\xi$ and `eff_e_`$\xi$ (where $e \in \mathcal{I}_{\Psi,\xi}$). This encoding exploits the syntactic constraints of STRIPS so it is more compact that the one previously proposed by Aineto *et al.* 2018 for learning STRIPS action models with classical planning. In more detail, if `pre_e_`$\xi$ holds it means that $e \in \mathcal{I}_{\Psi,\xi}$ is a *precondition* in $\xi$. If `pre_e_`$\xi$ and `eff_e_`$\xi$ holds it means that $e \in \mathcal{I}_{\Psi,\xi}$ is a *negative effect* in $\xi$ while if *pre_e_*$\xi$ does not hold but `eff_e_`$\xi$ holds, it means that $e \in \mathcal{I}_{\Psi,\xi}$ is a *positive effect* in $\xi$. Figure 2 shows the PDDL encoding of the `stack(?v1,?v2)` schema and our propositional representation for this same schema using the `pre_e_stack` and `eff_e_stack` fluents ($e \in \mathcal{I}_{\Psi,stack}$).

```
(:action stack
   :parameters (?v1 ?v2)
   :precondition (and (holding ?v1) (clear ?v2))
   :effect (and (not (holding ?v1)) (not (clear ?v2))
                (clear ?v1) (handempty) (on ?v1 ?v2)))


(pre_holding_v1_stack) (pre_clear_v2_stack)
(eff_holding_v1_stack) (eff_clear_v2_stack)
(eff_clear_v1_stack) (eff_handempty_stack) (eff_on_v1_v2_stack)
```

Figure 2: PDDL encoding of the `stack(?v1,?v2)` schema and our propositional representation for this same schema.

### 3.2 The sampling space

We define a *learning example* as a sequence $\mathcal{O} = \langle s_0^o, s_1^o \ldots, s_m^o \rangle$ of *partially observed states*, except for the initial state $s_0^o$ which is a *full state*. The set of predicates $\Psi$ and the set of objects $\Omega$ that shape the fluents $F$ is then deducible from $\mathcal{O}$. A partially observed state $s_i^o$, $1 \leq i \leq m$, is one in which $|s_i^o| < |F|$; i.e., a state in which at least a fluent of $F$ was not observed. Intermediate states can be *missing*, meaning that they are *unobserved*, so transiting between two consecutive observed states in $\mathcal{O}$ may require the execution of more than a single action ($\theta(s_i^o, \langle a_1, \ldots, a_k \rangle) = s_{i+1}^o$ (where $k \geq 1$ is unknown but finite). The minimal expression of a learning example must comprise at least two state observations, a full initial state $s_0^o$ and a partially observed state $s_m^o$ so $m \geq 1$.

To illustrate this Figure 3 shows a learning example that contains an initial state of the blocksworld where the robot hand is empty and three blocks (namely `blockA`, `blockB` and `blockC`) are on top of the table and clear. The second observation is a partially observed state in which `blockA` is on top of `blockB` and `blockB` on top of `blockC`.

```
(:predicates (on ?x ?y) (ontable ?x)
     (clear ?x) (handempty)
     (holding ?x))

(:objects blockA blockB blockC)

(:init (ontable blockA) (clear blockA)
       (ontable blockB) (clear blockB)
       (ontable blockC) (clear blockC)
       (handempty))

(:observation (on blockA blockB) (on blockB blockC))
```

Figure 3: Example of a two-state observationn for the learning STRIPS action models.

### 3.3 The domain-specific knowledge

Our approach is to introduce *domain-specific knowledge* in the form of *state-constraints* to restrict further the space of possible schemata. For instance, in the *blocksworld* one can argue that `on`($v_1$,$v_1$) and `on`($v_2$,$v_2$) will not appear in the $pre(\xi)$, $del(\xi)$ and $add(\xi)$ lists of an action schema $\xi$ because, in this specific domain, a block cannot be on top of itself. The notion of *state-constraint* is very general and has been used in different areas of AI and for different purposes. If we restrict ourselves to planning, *state-constraints* are abstractions for the compact specification of a set of states. For example, *state-constraints* in planning allow to specify the set of states where a given action is applicable, the set of states where a given *axiom* or *derived predicate* holds or the set of states that are considered goal states.

*State-invariants* is a kind of state-constraints useful for computing more compact state representations of a given planning problem [Helmert, 2009] and for making *satisfiability planning* or *backward search* more efficient [Rintanen, 2014; Alcázar and Torralba, 2015]. Given a classical planning problem $P = \langle F, A, I, G\rangle$, a *state-invariant* is a formula $\phi$ that holds at the initial state of a given classical planning problem, $I \models \phi$, and at every state $s$, built from $F$, that is reachable from $I$ by applying actions in $A$. For instance, Figure 4 shows five clauses that define *state-invariants* for the *blocksworld* planning domain.

$\forall x_1, x_2\ ontable(x_1) \leftrightarrow \neg on(x_1, x_2)$.
$\forall x_1, x_2\ clear(x_1) \leftrightarrow \neg on(x_2, x_1)$.
$\forall x_1, x_2, x_3\ \neg on(x_1, x_2) \vee \neg on(x_1, x_3)\ such\ that\ x_2 \neq x_3$.
$\forall x_1, x_2, x_3\ \neg on(x_2, x_1) \vee \neg on(x_3, x_1)\ such\ that\ x_2 \neq x_3$.
$\forall x_1, \ldots, x_n\ \neg(on(x_1, x_2) \wedge on(x_2, x_3) \wedge \ldots \wedge on(x_{n-1}, x_n) \wedge on(x_n, x_1))$.

Figure 4: Example of *state-invariants* for the *blocksworld* domain.

A *mutex* (mutually exclusive) is a *state-invariant* that takes the form of a binary clause and indicates a pair of different properties that cannot be simultaneously true [Kautz and Selman, 1999]. For instance in a three-block *blocksworld*, $\neg on(block_A, block_B) \vee \neg on(block_A, block_C)$ is a *mutex* because $block_A$ can only be on top of a single block.

A *domain invariant* is an instance-independent state-invariant, i.e. holds for any possible initial state and any possible set of objects. Therefore, if a given state $s$ holds $s \not\models \phi$ such that $\phi$ is a *domain invariant*, it means that $s$ is not a valid state. Domain invariants are often compactly defined as *lifted invariants* (also called *schematic* invariants [Rintanen, 2017]).

In this work we exploit *domain-specific knowledge* that is given as *schematic mutex*. We pay special attention to *schematic mutex* because they identify mutually exclusive *properties* of a given type of objects [Fox and Long, 1998] and because they enable (1) effective completion of partially observed states and (2) effectively pruning of inconsistent STRIPS action models. We define a *schematic mutex* as a $\langle p, q\rangle$ pair where both $p, q \in \mathcal{I}_{\Psi,\xi}$ are predicates that shape the preconditions or effects of a given action scheme $\xi$ and such that they satisfy the formulae $\neg p \vee \neg q$, considering that their corresponding variables are *universally quantified*. For instance, $holding(v_1)$ and $clear(v_1)$ from the *blocksworld* are *schematic mutex* while $clear(v_1)$ and $ontable(v_1)$ are not because $\forall v_1, \neg clear(v_1) \vee \neg ontable(v_1)$ does not hold for every possible *blocksworld* state.

# 4 Learning STRIPS action models from *schematic mutex* with classical planning

This section shows how to solve the $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi\rangle$ learning task with an off-the-shelf classical planner.

| ID | Action | New conditional effect |
|---|---|---|
| 1 | $\mathsf{insertPre}_{\mathsf{p},\xi}$ | $\{pre\_q\_\xi\} \rhd \{invalid\}$ |
| 2 | $\mathsf{insertEff}_{\mathsf{p},\xi}$ | $\{pre\_q\_\xi \wedge eff\_q\_\xi \wedge pre\_p\_\xi\} \rhd \{invalid\}$ |
| 3 | $\mathsf{insertEff}_{\mathsf{p},\xi}$ | $\{\neg pre\_q\_\xi \wedge eff\_q\_\xi \wedge \neg pre\_p\_\xi\} \rhd \{invalid\}$ |
| 4 | $\mathsf{apply}_{\xi,\omega}$ | $\{\neg pre\_p\_\xi \wedge eff\_p\_\xi \wedge$ $q(\omega) \wedge \neg pre\_q\_\xi\} \rhd \{invalid\}$ |
| 5 | $\mathsf{apply}_{\xi,\omega}$ | $\{\neg pre\_p\_\xi \wedge eff\_p\_\xi \wedge$ $q(\omega) \wedge pre\_q\_\xi \wedge \neg eff\_q\_\xi\} \rhd \{invalid\}$ |

Figure 5: Summary of the new conditional effects added to the classical planning compilation for the learning of STRIPS action models.

## 4.1 Completing partially observed states with *schematic mutex*

Our sampling space follows the *open world* assumption, i.e. what is not observed is considered unknown. Here we describe a pre-processing mechanism to add new knowledge that completes the states $\langle s_1^o \ldots, s_m^o\rangle$ that are partially observed in a $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi\rangle$ learning task using an input set of *schematic mutex* $\Phi$.

Let us define the production rule $p \rightarrow \neg q$ such that $\langle p, q\rangle$ is a *schematic mutex*. Given a *schematic mutex* $\langle p, q\rangle \in \Phi$ and a state observation $s_j^o \in \mathcal{O}$, $(1 \leq j \leq m)$ then $s_j^o$ can be safely completed adding the new literals $\neg q(\omega)$ that result from the unification of the corresponding production rule with $s_j^o$. $\omega \subseteq \Omega^{pars(q)}$ represents the objects that unify the variables in $q$ such that $\Omega^k$ is the $k$-th Cartesian power of $\Omega$. For instance, if the literal `holding(blockA)` is observed in a particular blocksword state and $\Phi$ contains the *schematic mutex* $\neg holding(v_1) \vee \neg clear(v_1)$, we can safely extend that state observation with literal `¬clear(blockA)` (despite this particular literal was actually unobserved).

## 4.2 Pruning inconsistent action models with *schematic mutex*

We could extend the classical planning compilation for the learning of STRIPS action models [Aineto *et al.*, 2018] to check the consistency of the *state-constraints* in $\Phi$ at every state traversed by a solution to the compiled problem. Unfortunately, checking arbitrary $\phi$ formulae is too expensive for current classical planners.

Instead, our approach is to define a mechanism to check *state-constraints* in the form of *schematic mutex*. To implement this checking mechanism we add new conditional effects to the *insert* and *apply* actions of the classical planning compilation. Figure 5 summarizes the new conditional effects added to the compilation and next, we describe them in detail:

1-3 For every *schematic mutex* $\langle p, q\rangle$ s.t. both $p$ and $q$ belong to $\in \mathcal{I}_{\Psi,\xi}$ one conditional effect is added to the $\mathsf{insertPre}_{\mathsf{p},\xi}$ actions to ban the insertion of two preconditions that are *schematic mutex*. Likewise, two conditional effects are added to the $\mathsf{insertEff}_{\mathsf{p},\xi}$ actions, one to ban the insertion of two positive effects that are *schematic mutex* and another one to ban two mutex negative effects.

4-5 For every *schematic mutex* $\langle p, q\rangle$ s.t. both $p$ and $q$ belong to $\in \mathcal{I}_{\Psi,\xi}$ two conditional effects are added to the

$\mathsf{apply}_{\xi,\omega}$ actions to ban positive effects that are inconsistent with an input observation (in $\mathsf{apply}_{\xi,\omega}$ actions the variables in $pars(\xi)$ are bounded to the objects in $\omega$ that appear in the same position).

In theory, conditional effects of the kind 4 and 5 are enough to guarantee that all the states traversed by a plan produced by the compilation are *consistent* with the input set of *schematic mutex* $\Phi$ (of course, provided that the input initial state $s_0^o$ is a valid state). In practice we include also conditional effects of the kind 1, 2 and 3 because they prune *invalid* action models at an earlier stage of the planning process (these effects extend the *insert* actions that always appear first in the solution plans).

The goals of the classical planning problem output by the original compilation are extended with the $\neg invalid$ literal to validate that only states *consistent* with the state constraints defined in $\Phi$ are traversed by solution plans. Remarkably, the $\neg invalid$ literal allows us also to define $\mathsf{apply}_{\xi,\omega}$ actions more compactly than in the original compilation by Aineto *et al.* 2018. Disjunctions are no longer required to code the possible preconditions of an action schema since they can now be encoded with conditional effects of this kind $\{pre\_p\_\xi \wedge \neg p(\omega)\} \rhd \{invalid\}$.

### 4.3 Compilation properties

**Lemma 1.** *Soundness. Any classical plan $\pi_\Lambda$ that solves $P_\Lambda$ produces a* STRIPS *model $\mathcal{M}'$ that solves the $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$ learning task.*

*Proof.* According to the $P_\Lambda$ compilation, once a given precondition or effect is inserted into the action model $\mathcal{M}$ it cannot be removed back. In addition, once the action model $\mathcal{M}$ is applied it cannot be *programmed*. In the compiled planning problem $P_\Lambda$, only $\mathsf{apply}_{\xi,\omega}$ actions can update the value of the state fluents $F$. This means that a state consistent with an observation $s_n^o$ can only be achieved executing an applicable sequence of $\mathsf{apply}_{\xi,\omega}$ actions that, starting in the corresponding initial state $s_0^o$, validates that every generated intermediate state $s_i$, s.t. $0 \leq i \leq n$, is consistent with the input state observations and *state-invariants*. This is exactly the definition of the solution condition for an action model $\mathcal{M}'$ to solve the $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$ learning task. □

**Lemma 2.** *Completeness. Any* STRIPS *model $\mathcal{M}'$ that solves the $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$ learning task can be computed with a classical plan $\pi_\Lambda$ that solves $P_\Lambda$.*

*Proof.* By definition $\mathcal{I}_{\Psi,\xi}$ fully captures the set of elements that can appear in a STRIPS action schema $\xi$ using predicates $\Psi$. In addition the $P_\Lambda$ compilation does not discard any possible action model $\mathcal{M}'$ definable within $\mathcal{I}_{\Psi,\xi}$ that satisfies the domain mutex in $\Phi$. This means that, for every STRIPS model $\mathcal{M}'$ that solves the $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$, we can build a plan $\pi_\Lambda$ that solves $P_\Lambda$ by selecting the appropriate $\mathsf{insertPre}_{\mathsf{p},\xi}$ and $\mathsf{insertEff}_{\mathsf{p},\xi}$ actions for *programming* the precondition and effects of the corresponding action model $\mathcal{M}'$ and then, selecting the corresponding $\mathsf{apply}_{\xi,\omega}$ actions that transform the initial state observation $s_0^o$ into the final state observation $s_m^o$. □

The size of the classical planning task $P_\Lambda$ output by our compilation depends on the arity of the given *predicates* $\Psi$, that shape the propositional state variables $F$, and the number of parameters of the action models, $|pars(\xi)|$. The larger these arities, the larger $|\mathcal{I}_{\Psi,\xi}|$. The size of the $\mathcal{I}_{\Psi,\xi}$ set is the term that dominates the compilation size because it defines the $pre\_e\_\xi/eff\_e\_\xi$ fluents, the corresponding set of *insert* actions, and the number of conditional effects in the $\mathsf{apply}_{\xi,\omega}$ actions. Note that *typing* can be used straightforward to constrain the FOL interpretations of $\Psi$ over the parameters $pars(\xi)$ which significantly reduces $|\mathcal{I}_{\Psi,\xi}|$ and hence, the size of the classical planning task output by the compilation.

Classical planners tend to preffer shorter solution plans, so our compilation may introduce a bias to $\Lambda = \langle \mathcal{M}, \mathcal{O}, \Phi \rangle$ learning tasks preferring solutions that are referred to action models with a shorter number of *preconditions/effects*. In more detail, all $\{pre\_e\_\xi, eff\_e\_\xi\}_{\forall e \in \mathcal{I}_{\Psi,\xi}}$ fluents are false at the initial state of our $P_\Lambda$ compilation so classical planners tend to solve $P_\Lambda$ with plans that require a shorter number of *insert* actions.

This bias could be eliminated defining a cost function for the actions in $P_\Lambda$ (e.g. *insert* actions have *zero cost* while $\mathsf{apply}_{\xi,\omega}$ actions have a *positive constant cost*). In practice we use a different approach to disregard the cost of *insert* actions because classical planners are not proficiency optimizing *plan cost* when there are zero-cost actions. Instead, our approach is to use a SAT-based planner [Rintanen, 2014] that can apply all actions for inserting preconditions in a single planning step (these actions do not interact). Further, the actions for inserting action effects are also applied in another single planning step. The plan horizon for programming any action model is then always bound to 2, which significantly reduces the planning horizon. The SAT-based planning approach is also convenient because its ability to deal with classical planning problems populated with dead-ends and because symmetries in the insertion of preconditions/effects into an action model do not affect to the planning performance.

## 5 Evaluation

This section evaluates the performance of our approach for learning STRIPS action models with different amounts of available input knowledge.

#### Reproducibility

The domains used in the evaluation are IPC domains that satisfy the STRIPS requirement [Fox and Long, 2003], taken from the PLANNING.DOMAINS repository [Muise, 2016]. We only used 1 learning examples for each learning task and we fixed the examples for all the experiments so that we can evaluate the impact of the different amount and source of the input knowledge in the quality of the learned models. All experiments are run on an Intel Core i5 3.10 GHz x 4 with 8 GB of RAM.

The classical planner we used to solve the instances that result from our compilations is the SAT-based plannerMADAGASCAR [Rintanen, 2014]. We used MADAGASCAR due to its ability to deal with planning instances populated with dead-ends [López *et al.*, 2015].

For the sake of reproducibility, the compilation source code, evaluation scripts, used benchmarks and input *state-invariants* are fully available at the repository *https://github.com/anonsub/*.

## 6 Related work

In *Inductive Logic Programming* it is common to make the hypothesis be consistent with the *background knowledge*, that is some form *deductive knowledge* apart from the examples [Muggleton and De Raedt, 1994].

*State-invariants* have also been previously used to improve the automatic construction of HTN planning model [Lotinac and Jonsson, 2016].

Our learning setting is related to the classical planning formulation where no action model is given [**?**]. This planning setting can can be seen as an scenario when the action model is *learned* from a single example that contains only two state observations: the initial state and the goals.

## 7 Conclusions

In some contexts it is however reasonable to assume that the action model is not learned from scratch, e.g. because some parts of the action model are known [Zhuo *et al.*, 2013; Sreedharan *et al.*, 2018; Pereira and Meneguzzi, 2018]. Our compilation is also flexible to this particular learning scenario. The known preconditions and effects are encoded setting the corresponding fluents $\{pre\_e\_\xi, eff\_e\_\xi\}_{\forall e \in \mathcal{I}_{\Psi,\xi}}$ to true in the initial state. Further, the corresponding insert actions, $\mathsf{insertPre}_{\mathsf{p},\xi}$ and $\mathsf{insertEff}_{\mathsf{p},\xi}$, become unnecessary and are removed from $A_\Lambda$, making the classical planning task $P_\Lambda$ easier to be solved. For example, suppose that the preconditions of the *blocksworld* action schema `stack` are known, then the initial state is extended with literals, `(pre_holding_v1_stack)` and `(pre_clear_v2_stack)` and the associated actions $\mathsf{insertPre}_{\mathsf{holding_v1,stack}}$ and $\mathsf{insertPre}_{\mathsf{clear_v2,stack}}$ can be safely removed from the $A_\Lambda$ action set without altering the *soundness* and *completeness* of the $P_\Lambda$ compilation.

## References


[Aineto *et al.*, 2018] Diego Aineto, Sergio Jiménez, and Eva Onaindia. Learning STRIPS action models with classical planning. In *International Conference on Automated Planning and Scheduling, (ICAPS-18)*, pages 399–407. AAAI Press, 2018.

[Alcázar and Torralba, 2015] Vidal Alcázar and Alvaro Torralba. A reminder about the importance of computing and exploiting invariants in planning. In *ICAPS*, pages 2–6. AAAI Press, 2015.

[Amir and Chang, 2008] Eyal Amir and Allen Chang. Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research*, 33:349–402, 2008.

[Fox and Long, 1998] Maria Fox and Derek Long. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research*, 9:367–421, 1998.

[Fox and Long, 2003] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[Helmert, 2009] Malte Helmert. Concise finite-domain representations for pddl planning tasks. *Artificial Intelligence*, 173(5-6):503–535, 2009.

[Kautz and Selman, 1999] Henry Kautz and Bart Selman. Unifying SAT-based and graph-based planning. In *IJCAI*, volume 99, pages 318–325, 1999.

[Kucera and Barták, 2018] Jirí Kucera and Roman Barták. LOUGA: learning planning operators using genetic algorithms. In *Pacific Rim Knowledge Acquisition Workshop, PKAW-18*, pages 124–138, 2018.

[López *et al.*, 2015] Carlos Linares López, Sergio Jiménez Celorrio, and Ángel García Olaya. The deterministic part of the seventh international planning competition. *Artificial Intelligence*, 223:82–119, 2015.

[Lotinac and Jonsson, 2016] Damir Lotinac and Anders Jonsson. Constructing hierarchical task models using invariance analysis. In *ECAI*, pages 1274–1282, 2016.

[McDermott *et al.*, 1998] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL – The Planning Domain Definition Language, 1998.

[Mourão *et al.*, 2012] Kira Mourão, Luke S. Zettlemoyer, Ronald P. A. Petrick, and Mark Steedman. Learning STRIPS operators from noisy and incomplete observations. In *Conference on Uncertainty in Artificial Intelligence, UAI-12*, pages 614–623, 2012.

[Muggleton and De Raedt, 1994] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994.

[Muise, 2016] Christian Muise. Planning.domains. *ICAPS system demonstration*, 2016.

[Pereira and Meneguzzi, 2018] Ramon Fraga Pereira and Felipe Meneguzzi. Heuristic approaches for goal recognition in incomplete domain models. *arXiv preprint arXiv:1804.05917*, 2018.

[Rintanen, 2014] Jussi Rintanen. Madagascar: Scalable planning with SAT. In *International Planning Competition, (IPC-2014)*, 2014.

[Rintanen, 2017] Jussi Rintanen. Schematic invariants by reduction to ground invariants. In *AAAI*, pages 3644–3650, 2017.

[Slaney and Thiébaux, 2001] John Slaney and Sylvie Thiébaux. Blocks world revisited. *Artificial Intelligence*, 125(1-2):119–153, 2001.

[Sohrabi *et al.*, 2016] Shirin Sohrabi, Anton V. Riabov, and Octavian Udrea. Plan recognition as planning revisited. In *International Joint Conference on Artificial Intelligence, (IJCAI-16)*, pages 3258–3264, 2016.

[Sreedharan *et al.*, 2018] Sarath Sreedharan, Tathagata Chakraborti, and Subbarao Kambhampati. Handling model uncertainty and multiplicity in explanations via model reconciliation. In *International Conference on Automated Planning and Scheduling, (ICAPS-18)*, pages 518–526, 2018.

[Yang *et al.*, 2007] Qiang Yang, Kangheng Wu, and Yunfei Jiang. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence*, 171(2-3):107–143, 2007.

[Zhuo and Kambhampati, 2013] Hankz Hankui Zhuo and Subbarao Kambhampati. Action-model acquisition from noisy plan traces. In *International Joint Conference on Artificial Intelligence, IJCAI-13*, pages 2444–2450, 2013.

[Zhuo *et al.*, 2013] Hankz Hankui Zhuo, Tuan Anh Nguyen, and Subbarao Kambhampati. Refining incomplete planning domain models through plan traces. In *International Joint Conference on Artificial Intelligence, IJCAI-13*, pages 2451–2458, 2013.

[Zhuo, 2015] Hankz Hankui Zhuo. Crowdsourced action-model acquisition for planning. In *National Conference on Artificial Intelligence, AAAI-15*, pages 3439–3446, 2015.