

# Learning STRIPS action models from *state-invariants*

Diego Aineto<sup>1</sup>, Sergio Jiménez<sup>1</sup>, Eva Onaindia<sup>1</sup>

<sup>1</sup>Departamento de Sistemas Informáticos y Computación. Universitat Politècnica de València. Valencia, Spain

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

## Abstract

This paper addresses the learning of action models from *deductive* knowledge with the form of *state-invariants* (i.e. logic formulae that specify constraints about the possible states of a given domain) to reduce the negative effects of insufficient input observations. Given an action model, state-of-the-art planners use it to infer *state-invariants* that reduce the planner search space and make the planning process more efficient [Helmert, 2009]. In this paper we follow the opposite direction and exploit *state-invariants* to learn a planning action model.

## 1 Introduction

The specification of planning action models is a complex process that limits, too often, the application of *model-based planning* to real-world tasks [Kambhampati, 2007]. The *machine learning* of action models can relieve the *knowledge acquisition bottleneck* of planning and nowadays, there exists a wide range of effective approaches for learning action models [Arora *et al.*, 2018]. Many of the most successful approaches for learning planning action models are however purely *inductive* [Yang *et al.*, 2007; Pasula *et al.*, 2007; Mourao *et al.*, 2010; Zhuo and Kambhampati, 2013], meaning that their performance depends on the *amount* and *quality* of the input examples (namely observations of plan executions generated by the aimed action model).

This paper addresses the learning of action models from *deductive* knowledge with the form of *state-invariants* (i.e. logic formulae that specify constraints about the possible states of a given domain) to reduce the negative effects of insufficient input observations. Given an action model, state-of-the-art planners use it to infer *state-invariants* that reduce the planner search space and make the planning process more efficient [Helmert, 2009]. In this paper we follow the opposite direction and exploit *state-invariants* to learn a planning action model.

Our approach is built on top of the *classical planning* compilation for the learning of STRIPS action models [Aineto *et al.*, 2018]. This compilation is flexible to different kinds of input knowledge (e.g., partially/fully observations of actions of plan executions as well as partially/fully observed intermediate states) and outputs an action model that is *consistent*

with the given input knowledge. In this paper we show that, in unfavorable scenarios where input observations are minimal (just an *initial state* and the *goals*), *state-invariant* help to learn better STRIPS models with the *classical planning* compilation.

## 2 Background

This section formalizes the *classical planning model* we follow in this work and the kind of *knowledge* that can be given as input to the task of learning STRIPS action models.

### 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  $\text{pre}(a) \in \mathcal{L}(F)$ , a set of effects  $\text{eff}(a) \in \mathcal{L}(F)$ , and a positive action cost  $\text{cost}(a)$ . 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  $\text{pre}(a) \subseteq s$ , 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  $\theta(s, a) = (s \setminus \neg \text{eff}(a)) \cup \text{eff}(a)$ . Subtracting the complement of  $\text{eff}(a)$  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  $\text{eff}^+(a) \in \text{eff}(a)$  while  $\text{eff}^-(a) \in \text{eff}(a)$  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  $\pi = \langle a_1, \dots, a_n \rangle$ , with  $|\pi| = n$  denoting its *plan length* and  $\text{cost}(\pi) = \sum_{a \in \pi} \text{cost}(a)$  its *plan cost*. The execution of  $\pi$  on the initial state of  $P$  induces a *trajectory*  $\tau(\pi, P) = \langle s_0, a_1, s_1, \dots, a_n, s_n \rangle$  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  $\text{pre}(a_c) \in \mathcal{L}(F)$  and a set of *conditional effects*  $\text{cond}(a_c)$ . Each conditional effect  $C \triangleright E \in \text{cond}(a_c)$  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  $\theta(s, a_c) = \{s \setminus \text{eff}_c(s, a) \cup \text{eff}_c(s, a)\}$  where  $\text{eff}_c(s, a)$  are the *triggered effects* resulting from the action application (conditional effects whose conditions hold in  $s$ ):

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

## 2.2 State-invariants

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 compactly specifying sets of states. For instance, *state-constraints* in planning allow to specify the set of states where a given action is applicable, the set of states where a given *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 [Helmert, 2009] or making *satisfiability planning* and *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  $A$ .

The formula  $\phi_{I,A}^*$  represents the *strongest invariant* and exactly characterizes the set of all states reachable from  $I$  with the actions in  $A$ . For instance Figure 1 shows five clauses that define the *strongest invariant* for the *blocksworld* planning domain [Slaney and Thiébaux, 2001]. There are infinitely many strongest invariants, but they are all logically equivalent, and computing the strongest invariant is PSPACE-hard (as hard as testing plan existence [Bylander, 1994]).

$$\begin{aligned} \forall x_1, x_2 \text{ ontable}(x_1) &\leftrightarrow \neg \text{on}(x_1, x_2). \\ \forall x_1, x_2 \text{ clear}(x_1) &\leftrightarrow \neg \text{on}(x_2, x_1). \\ \forall x_1, x_2, x_3 \neg \text{on}(x_1, x_2) \vee \neg \text{on}(x_1, x_3) &\text{ such that } x_2 \neq x_3. \\ \forall x_1, x_2, x_3 \neg \text{on}(x_2, x_1) \vee \neg \text{on}(x_3, x_1) &\text{ such that } x_2 \neq x_3. \\ \forall x_1, \dots, x_n \neg (\text{on}(x_1, x_2) \wedge \text{on}(x_2, x_3) \wedge \dots \wedge \text{on}(x_{n-1}, x_n) \wedge & \\ \text{on}(x_n, x_1)) &. \end{aligned}$$

Figure 1: *Strongest invariant* 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*,  $\phi_1 = \neg \text{on}(\text{block}_A, \text{block}_B) \vee \neg \text{on}(\text{block}_A, \text{block}_C)$  is a mutex because  $\text{block}_A$  can only be on top of a single block.

A *domain invariant* is an instance-independent invariant, i.e. holds for any possible initial state and set of objects. Therefore, if a given state  $s$  holds  $s \models \phi$  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 and others, 2017]. For instance,  $\phi_2 = \forall x : (\neg \text{handempty} \vee \neg \text{holding}(x))$ , is a *domain mutex* for the *blocksworld* because the robot hand is never empty and holding a block at the same time.

## 3 Learning STRIPS action models from state-invariants

We define the task of learning an action model from *state-invariants* as a tuple  $\Lambda = \langle P, \Phi, M \rangle$ , where:

- $P = \langle F, A[\cdot], I, G \rangle$ , is a *classical planning problem* where  $A[\cdot]$  is a set of actions s.t., the semantics of each action  $a \in A[\cdot]$  is unknown (i.e. the functions  $\rho$  and/or  $\theta$  of  $a$  are undefined).
- $\Phi$ , a set of *state-invariants* that constrain the set of possible states in the given domain.
- $M$  is the *space of possible action model* for the  $A[\cdot]$  actions. This set is the *full* space of action models, when learning from scratch, or a *partially specified action model*, when some fragments of the aimed action model are a priori known. A set of action models can be defined *explicitly*, enumerating all the models that belong to the set or *implicitly*, enumerating all the constraints that must satisfy any model that belongs to the set. A *partially specified* STRIPS action model is then a formalism for the *implicit* representation of a set of STRIPS schemes [Sreedharan et al., 2018].

A *solution* to a  $\Lambda = \langle P, \Phi, M \rangle$  learning task is a model  $M' \in M$  such that there exists a plan  $\pi$  satisfying that any state traversed in the trajectory  $\tau(\pi, P)$  satisfies all the *state-invariants* in  $\Phi$ . Inductive approaches for the learning planning action models compute an action model that maximizes some notion of *statistical consistency* over a set of observations of plan executions so output an action model in our case a solution to the addressed learning task is an action model that is *consistent* with the input knowledge.

Next we show that the set  $M$  of possible action models can be encoded as a set of propositional variables and a set of constraints over those variables. Then, we show how to exploit this encoding to solve a  $\Lambda = \langle P, \Phi, M \rangle$  learning task with an off-the-shelf classical planner.

### 3.1 A propositional encoding for the space of STRIPS action models

A STRIPS *action schema*  $\xi$  is defined by four lists: A list of *parameters*  $\text{pars}(\xi)$ , and three list of predicates (namely  $\text{pre}(\xi)$ ,  $\text{del}(\xi)$  and  $\text{add}(\xi)$ ) 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*  $\text{pars}(\xi)$ . The set of elements that

```

(: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.

can appear in  $pre(\xi)$ ,  $del(\xi)$  and  $add(\xi)$  of the STRIPS action schema  $\xi$  is given by 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(v1)` schemata,  $\mathcal{I}_{\Psi, pickup} = \{\text{handempty}, \text{holding}(v1), \text{clear}(v1), \text{ontable}(v1), \text{on}(v1, v1)\}$  while it contains eleven elements for the `stack(v1, v2)` schemata,  $\mathcal{I}_{\Psi, stack} = \{\text{handempty}, \text{holding}(v1), \text{holding}(v2), \text{clear}(v1), \text{clear}(v2), \text{ontable}(v1), \text{ontable}(v2), \text{on}(v1, v1), \text{on}(v1, v2), \text{on}(v2, v1), \text{on}(v2, v2)\}$ .

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

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. **Domain-specific constraints.** One can introduce domain-specific knowledge to constrain further the space of possible schemata. For instance, in the *blocksworld* one can argue that `on(v1, v1)` and `on(v2, v2)` 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. *State invariants* are also constraints of this kind.
3. **Observation constraints.** An observation  $\mathcal{O}(\tau)$  depicts *semantic knowledge* that constraints further the space of possible action schemata.

In this work we introduce a propositional encoding of the *preconditions*, *negative*, and *positive* effects of a STRIPS action schema  $\xi$  using 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 than the one previously proposed by Aineto *et al.* 2018 for learning classical planning action models. 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 with

$pre\_e\_ \xi$  and  $eff\_e\_ \xi$  fluents ( $e \in \mathcal{I}_{\Psi, stack}$ ).

### 3.2 Learning STRIPS action models with classical planning

Given a  $\Lambda = \langle P, \Phi, M \rangle$  where  $\Phi$  is a set of *domain mutex*  $\phi \in \Phi$ , we create a classical planning problem  $P' = \langle F', A', I, G \rangle$  such that:

- $F'$  extends  $F$  with a fluent *inconsistent*, to indicate whether an action model is *inconsistent* with the input *state-invariants*, a fluent *mode<sub>insert</sub>*, to indicate whether action models are being programmed, and the fluents for the propositional encoding of the corresponding space of STRIPS action models. This is a set of fluents of the type  $\{pre\_e\_ \xi, eff\_e\_ \xi\}_{\forall e \in \mathcal{I}_{\Psi, \xi}}$  such that  $e \in \mathcal{I}_{\Psi, \xi}$  is a single element from the set of FOL interpretations of predicates  $\Psi$  over the corresponding action parameters  $pars(\xi)$ .
- $A'$  replaces the actions in  $A$  with two types of actions.

1. Actions for *inserting* a *precondition*, *positive* effect or *negative* effect in  $\xi$  following the syntactic constraints of STRIPS models. In the particular case that  $M$  is a *partially specified model* then only the actions for inserting a possible *precondition* or *effect* are necessary.

- Actions which support the addition of a *precondition*  $p \in \Psi_\xi$  to the action model  $\xi$ . A precondition  $p$  is inserted in  $\xi$  when neither  $pre_p$ ,  $eff_p$  exist in  $\xi$ .

$$\begin{aligned} pre(\text{insertPre}_{p, \xi}) &= \{\neg pre_p(\xi), \neg eff_p(\xi), mode_{insert}\}, \\ cond(\text{insertPre}_{p, \xi}) &= \{\emptyset\} \triangleright \{pre_p(\xi)\}. \end{aligned}$$

- Actions which support the addition of a *negative* or *positive* effect  $p \in \Psi_\xi$  to the action model  $\xi$ .

$$\begin{aligned} pre(\text{insertEff}_{p, \xi}) &= \{\neg eff_p(\xi), mode_{insert}\}, \\ cond(\text{insertEff}_{p, \xi}) &= \{\emptyset\} \triangleright \{eff_p(\xi)\}. \end{aligned}$$

2. Actions for *applying* an action model  $\xi$  built by the *insert* actions and bounded to objects  $\omega \subseteq \Omega^{|pars(\xi)|}$  (where  $\Omega$  is the set of *objects* used to induce the fluents  $F$  by assigning objects in  $\Omega$  to the  $\Psi$  predicates and  $\Omega^k$  is the  $k$ -th Cartesian power of  $\Omega$ ). The action parameters,  $pars(\xi)$ , are bound to the objects in  $\omega$  that appear in the same position.

$$\begin{aligned} pre(\text{apply}_{\xi, \omega}) &= \{\emptyset\}, \\ cond(\text{apply}_{\xi, \omega}) &= \{pre_p(\xi) \wedge \neg p(\omega)\}_{\forall p \in \Psi_\xi}, \\ &\quad \{pre_p(\xi) \wedge eff_p(\xi)\} \triangleright \{\neg p(\omega)\}_{\forall p \in \Psi_\xi}, \\ &\quad \{\neg pre_p(\xi) \wedge eff_p(\xi)\} \triangleright \{p(\omega)\}_{\forall p \in \Psi_\xi}, \\ &\quad \{\emptyset\} \triangleright \{\neg mode_{insert}\}. \end{aligned}$$

## 4 Learning from *state-invariants* and observations of plan executions

Most of the approaches for learning planning action models compute an action model that maximizes some notion of *statistical consistency* over a set of observations of plan executions so output an action model that is *consistent* with the input knowledge.

### 4.1 The observation model

Given a planning problem  $P = \langle F, A, I, G \rangle$ , a plan  $\pi$  and a trajectory  $\tau(\pi, P)$ , we define the *observation of the trajectory* as an interleaved combination of actions and states that represents the observation from the execution of  $\pi$  in  $P$ . Formally,  $\mathcal{O}(\tau) = \langle s_0^o, a_1^o, s_1^o \dots, a_l^o, s_m^o \rangle$ ,  $s_0^o = I$ , and:

- The **observed actions** are consistent with  $\pi$ , which means that  $\langle a_1^o, \dots, a_l^o \rangle$  is a sub-sequence of  $\pi$ . The number of observed actions,  $l$ , ranges from 0 (fully unobserved action sequence) to  $|\pi|$  (fully observed action sequence).
- The **observed states**  $\langle s_0^o, s_1^o, \dots, s_m^o \rangle$  is a sequence of possibly *partially observable states*, except for the initial state  $s_0^o$ , which is fully observed. A partially observable state  $s_i^o$  is one in which  $|s_i^o| < |F|$ ; i.e., a state in which at least a fluent of  $F$  is not observable. Note that this definition also comprises the case  $|s_i^o| = 0$ , when the state is fully unobservable. Whatever the sequence of observed states of  $\mathcal{O}(\tau)$  is, it must be consistent with the sequence of states of  $\tau(\pi, P)$ , meaning that  $\forall i, s_i^o \subseteq s_i$ . The number of observed states,  $m$ , range from 1 (the initial state, at least), to  $|\pi| + 1$ , and each *observed* states comprises  $[1, |F|]$  fluents (the observation can still miss intermediate states that are *unobserved*).

We assume a bijective monotone mapping between actions/states of trajectories and observations [Ramírez and Geffner, 2009], thus also granting the inverse consistency relationship (the trajectory is a superset of the observation). Therefore, transiting between two consecutive observed states in  $\mathcal{O}(\tau)$  may require the execution of more than a single action ( $\theta(s_i^o, \langle a_1, \dots, a_k \rangle) = s_{i+1}^o$ , where  $k \geq 1$  is unknown but finite. In other words, having an input observation  $\mathcal{O}(\tau)$  does not imply knowing the actual length of  $\pi$ .

### 4.2 Extending the compilation to deal with *plan recognition as planning*

The work on *plan recognition as planning* usually assumes an observation model that is referred only to logs of executed actions. However, the approach applies also to more expressive observation models that consider state observations as well, like the observation model defined above, with a simple three-fold extension:

- One fluent  $\{validated_j\}_{0 \leq j \leq m}$  to point at every  $s_j^o \in \mathcal{O}(\tau)$  state observation. Two fluents,  $at_i$  and  $next_{i,i+1}$ ,  $1 \leq i \leq n$ , to iterate through the  $n$  observed actions of  $\tau$ . The former is used to ensure that actions are executed in the same order as they are observed in  $\tau$ . The latter is used to iterate to the next planning step when solving  $P_\Lambda$ .

- Adding  $at_1$  and  $\{next_{i,i+1}\}$ ,  $1 \leq i \leq n$  to the initial state and  $validated_m$  to every possible goal  $G \in G[\cdot]$  to constrain solution plans  $\pi^\top$  to be consistent with all the state observations.
- When the input plan trace contains observed actions, the extra conditional effects  $\{at_i, plan(name(a_i), \Omega^{ar(a_i)}, i)\} \triangleright \{\neg at_i, at_{i+1}\}_{\forall i \in [1, n]}$  are included in the  $apply_{\xi, \omega}$  actions to ensure that actions are applied in the same order as they appear in  $\tau$ .
- Actions for *validating* the partially observed state  $s_j \in \tau$ ,  $1 \leq j < m$ . These actions are also part of the postfix of the solution plan  $\pi_\Lambda$  and they are aimed at checking that the observable data of the input plan trace  $\tau$  follows after the execution of the apply actions.
- One  $validate_j$  action to constraint  $\pi^\top$  to be consistent with the  $s_j^o \in \mathcal{O}(\tau)$  input state observation, ( $1 \leq j \leq m$ ).

$$\begin{aligned} \text{pre}(\text{validate}_j) &= s_j^o \cup \{validated_{j-1}\}, \\ \text{cond}(\text{validate}_j) &= \{\emptyset\} \triangleright \{\neg validated_{j-1}, validated_j\}. \end{aligned}$$

*domain mutex* are useful to reduce the amount of applicable actions for programming a precondition or an effect for a given action schema. For example given the *domain mutex*  $\phi = (\neg f_1 \vee \neg f_2)$  such that  $f_1 \in F_v(\xi)$  and  $f_2 \in F_v(\xi)$ , we can redefine the corresponding programming actions for **removing** the *precondition*  $f_1 \in F_v(\xi)$  from the action schema  $\xi \in \mathcal{M}$  as:

## 5 Evaluation

## 6 Conclusions

## 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.
- [Arora *et al.*, 2018] Ankuj Arora, Humbert Fiorino, Damien Pellier, Marc Métivier, and Sylvie Pesty. A review of learning planning action models. *The Knowledge Engineering Review*, 2018.
- [Bylander, 1994] Tom Bylander. The computational complexity of propositional strips planning. *Artificial Intelligence*, 69(1-2):165–204, 1994.
- [Helmert, 2009] Malte Helmert. Concise finite-domain representations for pddl planning tasks. *Artificial Intelligence*, 173(5-6):503–535, 2009.

- [Kambhampati, 2007] Subbarao Kambhampati. Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models. In *National Conference on Artificial Intelligence, (AAAI-07)*, 2007.
- [Kautz and Selman, 1999] Henry Kautz and Bart Selman. Unifying SAT-based and graph-based planning. In *IJCAI*, volume 99, pages 318–325, 1999.
- [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.
- [Mourao *et al.*, 2010] Kira Mourao, Ronald PA Petrick, and Mark Steedman. Learning action effects in partially observable domains. In *ECAI*, pages 973–974. Citeseer, 2010.
- [Pasula *et al.*, 2007] Hanna M Pasula, Luke S Zettlemoyer, and Leslie Pack Kaelbling. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29:309–352, 2007.
- [Ramírez and Geffner, 2009] Miquel Ramírez and Hector Geffner. Plan recognition as planning. In *International Joint conference on Artificial Intelligence, (IJCAI-09)*, pages 1778–1783. AAAI Press, 2009.
- [Rintanen and others, 2017] Jussi Rintanen et al. Schematic invariants by reduction to ground invariants. In *AAAI*, pages 3644–3650, 2017.
- [Rintanen, 2014] Jussi Rintanen. Madagascar: Scalable planning with SAT. In *International Planning Competition, (IPC-2014)*, 2014.
- [Slaney and Thiébaux, 2001] John Slaney and Sylvie Thiébaux. Blocks world revisited. *Artificial Intelligence*, 125(1-2):119–153, 2001.
- [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.