

# Computing the *least-commitment* action model from state observations

Diego Aineto<sup>1</sup>, Sergio Jiménez<sup>1</sup>, Eva Onaindia<sup>1</sup> and , Blai Bonet<sup>2</sup>

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

<sup>2</sup>Departamento de Computación. Universidad Simón Bolívar. Caracas, Venezuela

{dieaigar,serjice,onaindia}@dsic.upv.es, bonet@usb.ve

## Abstract

## 1 Introduction

Given an input sequence of partially observed states, this paper formalizes the task of computing the *least-commitment* action model that is able to *explain* the given observation. This task is of interest because it allows the incremental learning of action models from arbitrary large sets of partial observations.

In addition, the paper introduces a new method to compute the *least-commitment* action model for an input sequence of partially observed states. The method assumes that action models are specified as STRIPS action schemata and it is built on top of off-the-shelf algorithms for *classical planning*.

## 2 Background

This section formalizes the *planning models* we use in the paper as well as the kind of state *observations* that are given as input for computing the *least-commitment* action model.

### 2.1 Classical planning with conditional effects

Let  $F$  be the set of *fluents* or *state variables* (propositional variables) 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 frame* is a tuple  $\Phi = \langle F, A \rangle$ , where  $F$  is a set of fluents and  $A$  is a set of *actions*. Each classical planning action  $a \in A$  has a precondition  $\text{pre}(a) \in \mathcal{L}(F)$  and a set of effects  $\text{eff}(a) \in \mathcal{L}(F)$ . 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*. The execution of  $\pi$  on the initial state  $I$  of  $P$  induces a *trajectory*  $\tau(\pi, s_0) = \langle s_0, a_1, s_1, \dots, a_n, s_n \rangle$  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*  $\tau(\pi, s_0)$  reaches a final state  $G \subseteq s_n$ , where all goal conditions are met. A solution plan is optimal iff its length is minimal.

An action  $a_c \in A$  with conditional effects is defined as 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  $\theta(s, a_c) = \{s \setminus \neg \text{eff}_c(s, a) \cup \text{eff}_c(s, a)\}$  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 The observation model

Given a classical planning problem  $P = \langle F, A, I, G \rangle$ , a plan  $\pi$  and a trajectory  $\tau(\pi, s_0)$ , we define the *observation of the trajectory* as a sequence of partial states that results from observing the execution of  $\pi$  on  $I$ . Formally,  $\mathcal{O}(\tau) = \langle s_0^o, s_1^o, \dots, s_m^o \rangle$  where  $s_0^o = I$ .

A *partial state*  $s_i^o$ ,  $0 < i < m$ , 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  $\tau(\pi, s_0)$ , meaning that  $\forall i, s_i^o \subseteq s_i$ . In practice, the number of observed states  $m$ , ranges from 1 (the initial state, at least), to  $|\pi| + 1$ , and the observed intermediate states will comprise a number of fluents between  $[1, |F|]$ .

We are assuming then that there is a *bijective monotone mapping* between 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  $\mathcal{O}(\tau)$  does not imply knowing the actual length of  $\pi$ .

**Definition 1 (Plan explanation)** *Given a classical planning problem  $P$  and an observation  $\mathcal{O}(\tau)$ , a plan  $\pi$  explains  $\mathcal{O}(\tau)$  (denoted  $\pi \mapsto \mathcal{O}(\tau)$ ) iff  $\pi$  is a solution for  $P$  that is consistent with the state trajectory constraints imposed by the sequence of partial states  $\mathcal{O}(\tau)$ .*

If  $\pi$  is also optimal, we say that  $\pi$  is the *best explanation* for the input observation  $\mathcal{O}(\tau)$ .

### 3 Computing the least-commitment action model from observations

First, this section formalizes the notion of the *least-commitment* action model that is able to *explain* an observation. Next, the section describes our approach to compute such model.

#### 3.1 The least-commitment action model

The task of computing the *least-commitment* action model from a sequence of state observations is defined as  $\langle \Phi, \mathcal{O}(\tau) \rangle$ :

- $\Phi = \langle F, A[\cdot] \rangle$  is a *classical planning frame* where the semantics of each action  $a \in A[\cdot]$  is unknown; i.e. the corresponding  $\langle \rho, \theta \rangle$  functions are undefined. We say that an *action model*  $\mathcal{M}$  is a definition of the  $\langle \rho, \theta \rangle$  functions of every action in  $A[\cdot]$ .
- $\mathcal{O}(\tau)$  is a sequence of partial states that results from the partial observation of a trajectory  $\tau(\pi, s_0)$  within the *classical planning frame*  $\Phi$ .

Given a *classical planning frame*  $\Phi = \langle F, A[\cdot] \rangle$  and an observation  $\mathcal{O}(\tau) = \langle s_0^o, s_1^o \dots, s_m^o \rangle$  let  $P_{\mathcal{O}}$  be the classical planning problem  $P_{\mathcal{O}} = \langle F, A[\cdot], s_0^o, s_m^o \rangle$ .

**Definition 2 (Model explanation)** *A model  $\mathcal{M}$  explains an observation  $\mathcal{O}(\tau)$  iff, when the  $\langle \rho, \theta \rangle$  functions of the actions in  $P_{\mathcal{O}}$  are given by  $\mathcal{M}$ , there exists a solution plan for  $P_{\mathcal{O}}$  that is the best explanation for  $\mathcal{O}(\tau)$ .*

**Definition 3 (The least-commitment action model)** *Given a  $\langle \Phi, \mathcal{O}(\tau) \rangle$  task (and let  $M$  be the set of action models that represents the full space of possible action models for the actions in  $A[\cdot] \in \Phi$ ), the least-commitment action model is the largest subset of models  $M^* \subseteq M$  such that every model  $\mathcal{M} \in M^*$  explains the input observation.*

This work focuses on the particular task of computing the *least-commitment* action model when action models are specified as STRIPS action schemata.

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

#### 3.2 The space of STRIPS action models

A STRIPS *action schema*  $\xi$  is defined by four lists: A list of *parameters*  $pars(\xi)$ , and three list 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 given by FOL interpretations of  $\Psi$  over the parameters  $pars(\xi)$ . We denote this set of FOL interpretations as  $\mathcal{I}_{\Psi, \xi}$ . For instance, in the *blocksworld* the  $\mathcal{I}_{\Psi, \xi}$  set contain 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  $\xi$ , the 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( $v_1, v_1$ )` 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. *State invariants* are also constraints of this kind [Fox and Long, 1998].
3. *Observation constraints.* An observations  $\mathcal{O}(\tau)$  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 is more compact than the one previously proposed by Aineto *et al.* 2018. In more detail, 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  $eff\_e\_ \xi$

```

(Kpre_holding_v1_stack) (Kpre_clear_v2_stack)
(K¬pre_holding_v2_stack) (K¬pre_clear_v1_stack)
(K¬pre_hanempty_stack)
(K¬pre_ontable_v1_stack) (K¬pre_ontable_v2_stack)
(K¬pre_on_v1_v2_stack) (K¬pre_on_v2_v1_stack)
(K¬pre_on_v1_v1_stack) (K¬pre_on_v2_v2_stack)

(Keff_holding_v2_stack) (Keff_clear_v2_stack)
(Keff_clear_v2_stack) (Keff_on_v2_v2_stack)
(Keff_on_v1_v1_stack) (Keff_on_v2_v1_stack)

```

Figure 2: *Partially specified* STRIPS model that represents of a set of action models for the `stack (?v1, ?v2)` schema.

holds, it means that  $e \in \mathcal{I}_{\Psi, \xi}$  is a positive effect in  $\xi$ . Figure 1 shows the PDDL encoding of the `stack (?v1, ?v2)` schema and our propositional representation for this same schema.

### 3.3 Partially specified STRIPS action models

A set of action models can be defined *explicitely*, by enumerating all the models that belong to the set or *implicitely*, by enumerating the constraints that must satisfy any model that belongs to the set. Inspired by the notion of *incomplete (annotated) model* [Sreedharan et al., 2018], we introduce here *partially specified* STRIPS action models, a formalism for the *implicit* representation of a set of STRIPS schema.

We show now that extending to the *knowledge level* our propositional encoding of STRIPS action schemes, we can compactly represent a set of STRIPS schema. The extension defines, for each proposition  $pre\_e.\xi$ , with two propositions  $Kpre\_e.\xi$  and  $K¬pre\_e.\xi$ , meaning that is *known* that  $e \in \mathcal{I}_{\Psi, \xi}$  is a precondition of  $\xi$  and that is *known* that  $e \in \mathcal{I}_{\Psi, \xi}$  is not a precondition of  $\xi$ . Likewise for each  $eff\_e.\xi$  proposition we define these two,  $Keff\_e.\xi$  and  $K¬eff\_e.\xi$ , meaning that is *known* that  $e \in \mathcal{I}_{\Psi, \xi}$  is an effect of  $\xi$  and that is *known* that  $e \in \mathcal{I}_{\Psi, \xi}$  is not an effect of  $\xi$ . With respect to this formalization, the *full space* of possible STRIPS schemas for  $\xi$  is compactly represented as  $\bigcap_{e \in \mathcal{I}_{\Psi, \xi}} \neg Kpre\_e.\xi \wedge \neg K¬pre\_e.\xi \wedge \neg Keff\_e.\xi \wedge \neg K¬eff\_e.\xi$ .

Figure 2 shows a *partially specified* STRIPS model that represents of a set of action models for the `stack (?v1, ?v2)` schema. This *partially specified* STRIPS model represents a set of  $2^5$  models since (1), the actual five effects of the action are *unknown*, (2) preconditions are known to appear as in the actual definition of the `stack (?v1, ?v2)` schema for the blocksworld and (3), it is *known* that the preconditions and effects that are not part of the actual definition of the `stack (?v1, ?v2)` schema are not part of this *partially specified* STRIPS model.

#### Definition 4 (Partially specified model explanation)

### 3.4 Computing the least-commitment model via classical planning

Inspired by the *classical planning compilation*  $K_0$  for conformant planning [Palacios and Geffner, 2009], this section shows that we can build a *classical planning problem*  $P = \langle F', A', I', G' \rangle$  whose solution induces the *least-commitment* action model for an observation  $\mathcal{O}(\tau)$ :

- The set of fluents  $F'$  extends  $F$  with two new sets of fluents:

- $\{test_j\}_{1 \leq j \leq m}$ , indicating the state observation  $s_j \in \mathcal{O}(\tau)$  where the action model is validated
- The *knowledge level* fluents  $Kpre\_e.\xi$ ,  $K¬pre\_e.\xi$ ,  $Keff\_e.\xi$  and  $K¬eff\_e.\xi$  encoding the space of possible *partially specified* action models.

- The set of actions  $A'$  contains now actions of three different kinds:

- Actions for *committing*  $pre\_e.\xi$  to a positive/negative value. Similar actions are also defined for *committing*  $eff\_e.\xi$  to a positive/negative value but the value of  $eff\_e.\xi$  can only be committed once the value of the corresponding  $pre\_e.\xi$  is committed (i.e. once either  $Kpre\_e.\xi$  or  $K¬pre\_e.\xi$  holds in the current state).

$$\begin{aligned}
pre(commit\_pre\_e.\xi) &= \{mode_{commit}, \\
&\quad \neg Kpre\_e.\xi, \neg K¬pre\_e.\xi\}, \\
cond(commit\_pre\_e.\xi) &= \{\emptyset\} \triangleright \{Kpre\_e.\xi\}.
\end{aligned}$$

$$\begin{aligned}
pre(commit\_¬pre\_e.\xi) &= \{mode_{commit}, \\
&\quad \neg Kpre\_e.\xi, \neg K¬pre\_e.\xi\}, \\
cond(commit\_¬pre\_e.\xi) &= \{\emptyset\} \triangleright \{K¬pre\_e.\xi\}.
\end{aligned}$$

- Actions for *validating* that committed models explain the  $s_j$  observed states,  $0 \leq j < m$ .

$$\begin{aligned}
pre(validate_j) &= s_j \cup \{test_{j-1}\}, \\
cond(validate_j) &= \{\emptyset\} \triangleright \{\neg test_{j-1}, test_j\}, \\
&\quad \{mode_{commit}\} \triangleright \{\neg mode_{commit}, mode_{val}\}.
\end{aligned}$$

- *Editable* actions whose semantics is given by the value of the *knowledge level* fluents ( $Kpre\_e.\xi$ ,  $K¬pre\_e.\xi$ ,  $Keff\_e.\xi$  and  $K¬eff\_e.\xi$ ) at the current state. Figure 3 shows the PDDL encoding of an *editable* `stack (?v1, ?v2)` schema. This editable schema behaves exactly as the original PDDL schema defined in Figure 1 when the set of fluents

$$\begin{aligned}
pre(editable_\xi) &= \{\neg K¬pre\_e.\xi \implies e\}_{e \in \mathcal{I}_{\Psi, \xi}} \\
cond(editable_\xi) &= \{Kpre\_e.\xi, \neg K¬eff\_e.\xi\} \triangleright \{\neg e\}_{e \in \mathcal{I}_{\Psi, \xi}}, \\
&\quad \{K¬pre\_e.\xi, \neg K¬eff\_e.\xi\} \triangleright \{e\}_{e \in \mathcal{I}_{\Psi, \xi}}.
\end{aligned}$$

- The new initial state  $I' = I \cup \{mode_{commit}\}$  while the new goals are  $G' = s_m \cup \{test_m\}$ .

### 3.5 Compilation properties

## 4 Evaluation

## 5 Conclusions

Related work [Stern and Juba, 2017].

```

(:action stack
:parameters (?o1 - object ?o2 - object)
:precondition
  (and (or (Knotpre_on_v1_v1_stack) (on ?o1 ?o1))
        (or (Knotpre_on_v1_v2_stack) (on ?o1 ?o2))
        (or (Knotpre_on_v2_v1_stack) (on ?o2 ?o1))
        (or (Knotpre_on_v2_v2_stack) (on ?o2 ?o2))
        (or (Knotpre_ontable_v1_stack) (ontable ?o1))
        (or (Knotpre_ontable_v2_stack) (ontable ?o2))
        (or (Knotpre_clear_v1_stack) (clear ?o1))
        (or (Knotpre_clear_v2_stack) (clear ?o2))
        (or (Knotpre_holding_v1_stack) (holding ?o1))
        (or (Knotpre_holding_v2_stack) (holding ?o2))
        (or (Knotpre_handempty_stack) (handempty)))
:effect (and
  (when (and (Kpre_on_v1_v1_stack) (not (Knot_eff_on_v1_v1_stack))) (not (on ?o1 ?o1)))
  (when (and (Kpre_on_v1_v2_stack) (not (Knot_eff_on_v1_v2_stack))) (not (on ?o1 ?o2)))
  (when (and (Kpre_on_v2_v1_stack) (not (Knot_eff_on_v2_v1_stack))) (not (on ?o2 ?o1)))
  (when (and (Kpre_on_v2_v2_stack) (not (Knot_eff_on_v2_v2_stack))) (not (on ?o2 ?o2)))
  (when (and (Kpre_ontable_v1_stack) (not (Knot_eff_ontable_v1_stack))) (not (ontable ?o1)))
  (when (and (Kpre_ontable_v2_stack) (not (Knot_eff_ontable_v2_stack))) (not (ontable ?o2)))
  (when (and (Kpre_clear_v1_stack) (not (Knot_eff_clear_v1_stack))) (not (clear ?o1)))
  (when (and (Kpre_clear_v2_stack) (not (Knot_eff_clear_v2_stack))) (not (clear ?o2)))
  (when (and (Kpre_holding_v1_stack) (not (Knot_eff_holding_v1_stack))) (not (holding ?o1)))
  (when (and (Kpre_holding_v2_stack) (not (Knot_eff_holding_v2_stack))) (not (holding ?o2)))
  (when (and (Kpre_handempty_stack) (not (Knot_eff_handempty_stack))) (not (handempty)))
  (when (and (Knot_pre_on_v1_v1_stack) (not (Knot_eff_on_v1_v1_stack))) (on ?o1 ?o1))
  (when (and (Knot_pre_on_v1_v2_stack) (not (Knot_eff_on_v1_v2_stack))) (on ?o1 ?o2))
  (when (and (Knot_pre_on_v2_v1_stack) (not (Knot_eff_on_v2_v1_stack))) (on ?o2 ?o1))
  (when (and (Knot_pre_on_v2_v2_stack) (not (Knot_eff_on_v2_v2_stack))) (on ?o2 ?o2))
  (when (and (Knot_pre_ontable_v1_stack) (not (Knot_eff_ontable_v1_stack))) (ontable ?o1))
  (when (and (Knot_pre_ontable_v2_stack) (not (Knot_eff_ontable_v2_stack))) (ontable ?o2))
  (when (and (Knot_pre_clear_v1_stack) (not (Knot_eff_clear_v1_stack))) (clear ?o1))
  (when (and (Knot_pre_clear_v2_stack) (not (Knot_eff_clear_v2_stack))) (clear ?o2))
  (when (and (Knot_pre_holding_v1_stack) (not (Knot_eff_holding_v1_stack))) (holding ?o1))
  (when (and (Knot_pre_holding_v2_stack) (not (Knot_eff_holding_v2_stack))) (holding ?o2))
  (when (and (Knot_pre_handempty_stack) (not (Knot_eff_handempty_stack))) (handempty)))

```

[Stern and Juba, 2017] Roni Stern and Brendan Juba. Efficient, safe, and probably approximately complete learning of action models. In *International Joint Conference on Artificial Intelligence, (IJCAI-17)*, pages 4405–4411, 2017.

Figure 3: PDDL encoding of the editable version of the stack(?v1, ?v2) schema.

## 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.
- [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.
- [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.
- [Palacios and Geffner, 2009] Héctor Palacios and Héctor Geffner. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research*, 35:623–675, 2009.
- [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.
- [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.