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

Diego Aineto[1] , **Sergio Jiménez**[1] , **Eva Onaindia**[1] and , **Blai Bonet**[2]

[1]Departamento de Sistemas Informáticos y Computación. Universitat Politècnica de València. Valencia, Spain
[2]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 *conformant 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 $\mathsf{pre}(a) \in \mathcal{L}(F)$ and a set of effects $\mathsf{eff}(a) \in \mathcal{L}(F)$. 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 $\mathsf{pre}(a) \subseteq s$, 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* π is an action sequence $\pi = \langle a_1, \ldots, a_n\rangle$, with $|\pi| = n$ denoting its *plan length*. The execution of π on the initial state I of P induces a *trajectory* $\tau(\pi, s_0) = \langle s_0, a_1, s_1, \ldots, a_n, s_n\rangle$ 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 π 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 $\mathsf{pre}(a_c) \in \mathcal{L}(F)$ 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 The observation model

Given a classical planning problem $P = \langle F, A, I, G\rangle$, a plan π 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 π on I. Formally, $\mathcal{O}(\tau) = \langle s_0^o, s_1^o \ldots, s_m^o\rangle$ 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 ($\theta(s_i^o, \langle a_1, \ldots, a_k\rangle) = s_{i+1}^o$, where $k \geq 1$ is unknown but finite. In other words, having $\mathcal{O}(\tau)$ does not imply knowing the actual length of π.

Definition 1 (Explaning a $\mathcal{O}(\tau)$ observation) *Given a* classical planning problem P *and a sequence of partial states* $\mathcal{O}(\tau)$*, a plan* π explains $\mathcal{O}(\tau)$ *(denoted* $\pi \mapsto \mathcal{O}(\tau)$*) iff* π *is a solution for* P consistent *with* $\mathcal{O}(\tau)$.

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

Given a *classical planning frame* $\Phi = \langle F, A[\cdot]\rangle$ and a sequence of partial states $\mathcal{O}(\tau) = \langle s_0^o, s_1^o \ldots, s_m^o\rangle$, we can build the classical planning problem $P_{\mathcal{O}} = \langle F, A[\cdot], s_0^o, s_m^o\rangle$. We say that an action model $\mathcal{M}$ is a definition of the $\langle\rho, \theta\rangle$ functions of every action in $A[\cdot]$. Further we say that a model $\mathcal{M}$ *explains* a sequence of observations $\mathcal{O}(\tau)$ 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 explains $\mathcal{O}(\tau)$.

2.3 Conformant planning

Conformant planning is planning with incomplete information about the initial state, no sensing, and validating that goals are achieved with certainty (despite the uncertainty of the initial state) [Goldman and Boddy, 1996; Smith and Weld, 1998; Bonet and Geffner, 2000].

Syntactically, conformant planning problems are expressed in compact form through a set of state variables. A *conformant planning problem* can be defined as a tuple $P_c = \langle F, A, \Upsilon, G\rangle$ where F, A and G are the set of *fluents*, *actions* and *goals* (as previously defined for *classical planning*). Now Υ is a set of clauses over literals $l = f$ or $l = \neg f$ (for $f \in F$) that define the set of possible initial states.

A solution to a conformant planning problem is an action sequence that maps each possible initial state into a goal state. More precisely, an action sequence $\pi = \langle a_1, \ldots, a_n\rangle$ is a *conformant plan* for P_c iff, for each possible *trajectory* $\tau(\pi, s_0) = \langle s_0, a_1, s_1, \ldots, a_n, s_n\rangle$ s.t. s_0 is a valuation of the fluents in F that satisfies Υ, then $\tau(\pi, s_0)$ reaches a final state $G \subseteq s_n$.

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

First, this section formalizes the notion of the *least-commitment* action model that is able to *explain* a sequence of partially observed states. Next, the section describes our approach to compute such model via *conformant planning*.

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.
- $\mathcal{O}(\tau)$ is a sequence of partial states that results from the partial observation of a trajectory $\tau(\pi, s_0)$ defined within the *classical planning frame* Φ.

Before formalizing the solution to this task, i.e. the *least-commitment* action model, we introduce several necessary definitions.

Definition 2 (Model Space) *Given a* classical planning frame $\Phi = \langle F, A[\cdot]\rangle$ *the model space* M *is the set of possible models for the actions in* $A[\cdot]$ *such that: (1), any model* $\mathcal{M} \in M$ *is a definition of the* $\langle\rho, \theta\rangle$ *functions of every action in* $A[\cdot]$ *and (2), for every* $\mathcal{M} \in M$ *the* $\langle\rho, \theta\rangle$ *functions are defined in the set of state variables* F.

Now, we define a *partially specified action model* inspired by the notion of *incomplete (annotated) model* [Sreedharan *et al.*, 2018].

Definition 3 (Partially specified action model) *A* partially specified action model *is a subset of models in a given* model space M.

If the *partially specified action model* is a singleton, it represents a *fully specified action model*. On the other hand, if its size is $|M|$ the *partially specified action model* represents the full model space.

Now we are ready to define the *least-commitment* action model for an observation $\mathcal{O}(\tau)$.

Definition 4 (The *least-commitment* action model) *Given a* $\langle\Phi, \mathcal{O}(\tau)\rangle$ *task and the* partially specified action model M *that represents the full space of possible action models for the actions in* $A[\cdot] \in \Phi$*, then the* least-commitment *action model is another* partially specified action model *that represents the largest subset of models* $M^* \subseteq M$ *such that every model* $\mathcal{M} \in M^*$ explains *the input observation.*

3.2 The space of STRIPS action models

Despite previous definitions are general, this work focuses on the particular kind of action models that are specified as STRIPS action schemata.

A STRIPS *action schema* ξ 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 Ψ 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 ξ is given by FOL interpretations of Ψ 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 ξ, the space of possible STRIPS schemata is bounded by constraints of three kinds:

```
(: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.

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 $2^{2\times|\mathcal{I}_{\Psi,\xi}|}$.
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 $\texttt{on}(v_1,v_1)$ and $\texttt{on}(v_2,v_2)$ will not appear in the $pre(\xi)$, $del(\xi)$ and $add(\xi)$ lists of an action schema ξ because, in this specific domain, a block cannot be on top of itself. *Typing constraints* and *state invariants* are also constraints of this kind [Fox and Long, 1998].
3. *Observation constraints*. A sequence of state observations $\mathcal{O}(\tau)$ depicts *semantic knowledge* that constraints further the space of possible action schemata.

3.3 A propositional encoding for STRIPS action models

With this regard, we implement a propositional encoding of the *preconditions*, *negative*, and *positive* effects of a STRIPS action schema ξ using only fluents of two kinds $\texttt{pre\_e\_}\xi$ and $\texttt{eff\_e\_}\xi$ (where $e \in \mathcal{I}_{\Psi,\xi}$). This encoding exploits the syntactic constraints of STRIPS so, if $\texttt{pre\_e\_}\xi$ and $\texttt{eff\_e\_}\xi$ holds it means that $e \in \mathcal{I}_{\Psi,\xi}$ is a negative effect in ξ while if pre_e_ξ does not hold but $\texttt{eff\_e\_}\xi$ holds, it means that $e \in \mathcal{I}_{\Psi,\xi}$ is a positive effect in ξ.

Figure 1 shows the PDDL encoding of the `stack(?v1,?v2)` schema and our propositional representation for this same schema. There is a total number of $2^{2\times|11|} = 4,194,304$ different models for that schema.

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

Inspired by the *classical planning compilation* K_{s_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
  - Fluents $K\mathsf{pre\_}e\_\xi$, $K\neg\mathsf{pre\_}e\_\xi$, $K\mathsf{eff\_}e\_\xi$ and $K\neg\mathsf{eff\_}e\_\xi$ encoding the *knowledge level* representation of the space of possible STRIPS action models.
- The set of actions A' contains now actions of three different kinds:
  - Actions for *committing* $\texttt{pre\_e\_}\xi$ to a positive/negative value. Similar actions are also defined for *committing* $\texttt{eff\_e\_}\xi$ to a positive/negative value but the value of $\texttt{eff\_e\_}\xi$ can only be committed once the value of the corresponding $\texttt{pre\_e\_}\xi$ is committed (i.e. once either $\mathsf{Kpre\_e\_}\xi$ or $\mathsf{K}\neg\mathsf{pre\_e\_}\xi$ holds in the current state).

$$\begin{aligned}\mathsf{pre}(\mathsf{commit}\top\mathsf{\_pre\_e\_}\xi) =&\{mode_{commit},\\ &\neg\mathsf{Kpre\_e\_}\xi, \neg\mathsf{K}\neg\mathsf{pre\_e\_}\xi\},\\ \mathsf{cond}(\mathsf{commit}\top\mathsf{\_pre\_e\_}\xi) =&\{\emptyset\} \rhd \{\mathsf{Kpre\_e\_}\xi\}.\end{aligned}$$

$$\begin{aligned}\mathsf{pre}(\mathsf{commit}\bot\mathsf{\_pre\_e\_}\xi) =&\{mode_{commit},\\ &\neg\mathsf{Kpre\_e\_}\xi, \neg\mathsf{K}\neg\mathsf{pre\_e\_}\xi\},\\ \mathsf{cond}(\mathsf{commit}\bot\mathsf{\_pre\_e\_}\xi) =&\{\emptyset\} \rhd \{\mathsf{K}\neg\mathsf{pre\_e\_}\xi\}.\end{aligned}$$

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

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

  - *Editable* actions whose semantics is given by the value of the *knowledge level* fluents ($K\mathsf{pre\_}e\_\xi$, $K\neg\mathsf{pre\_}e\_\xi$, $K\mathsf{eff\_}e\_\xi$ and $K\neg\mathsf{eff\_}e\_\xi$) at the current state. Figure 2 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 `(Kpre_holding_v1_stack) (Kpre_clear_v2_stack) (Keff_holding_v1_stack) (Keff_clear_v2_stack) (Keff_clear_v1_stack) (Keff_handempty_stack) (Keff_on_v1_v2_stack)` hold at the current state. Formally, given an operator schema $\xi \in \mathcal{M}$ its *editable* version is:

$$\begin{aligned}\mathsf{pre}(\mathsf{editable}_\xi) =&\{\neg K\neg\mathsf{pre\_}e\_\xi \implies e\}_{\forall e\in\mathcal{I}_{\Psi,\xi}}\\ \mathsf{cond}(\mathsf{editable}_\xi) =&\{K\mathsf{pre\_}e\_\xi, \neg K\neg\mathsf{eff\_}e\_\xi\} \rhd \{\neg e\}_{\forall e\in\mathcal{I}_{\Psi,\xi}},\\ &\{K\neg\mathsf{pre\_}e\_\xi, \neg K\neg\mathsf{eff\_}e\_\xi\} \rhd \{e\}_{\forall e\in\mathcal{I}_{\Psi,\xi}}.\end{aligned}$$

- 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)(Keff_on_v1_v1_stack)) (not (on ?o1 ?o1)))
   (when (and (Kpre_on_v1_v2_stack)(Keff_on_v1_v2_stack)) (not (on ?o1 ?o2)))
   (when (and (Kpre_on_v2_v1_stack)(Keff_on_v2_v1_stack)) (not (on ?o2 ?o1)))
   (when (and (Kpre_on_v2_v2_stack)(Keff_on_v2_v2_stack)) (not (on ?o2 ?o2)))
   (when (and (Kpre_ontable_v1_stack)(Keff_ontable_v1_stack)) (not (ontable ?o1)))
   (when (and (Kpre_ontable_v2_stack)(Keff_ontable_v2_stack)) (not (ontable ?o2)))
   (when (and (Kpre_clear_v1_stack)(Keff_clear_v1_stack)) (not (clear ?o1)))
   (when (and (Kpre_clear_v2_stack)(Keff_clear_v2_stack)) (not (clear ?o2)))
   (when (and (Kpre_holding_v1_stack)(Keff_holding_v1_stack)) (not (holding ?o1)))
   (when (and (Kpre_holding_v2_stack)(Keff_holding_v2_stack)) (not (holding ?o2)))
   (when (and (Kpre_handempty_stack)(Keff_handempty_stack)) (not (handempty)))
   (when (and (Knot_pre_on_v1_v1_stack)(Keff_on_v1_v1_stack)) (on ?o1 ?o1))
   (when (and (Knot_pre_on_v1_v2_stack)(Keff_on_v1_v2_stack)) (on ?o1 ?o2))
   (when (and (Knot_pre_on_v2_v1_stack)(Keff_on_v2_v1_stack)) (on ?o2 ?o1))
   (when (and (Knot_pre_on_v2_v2_stack)(Keff_on_v2_v2_stack)) (on ?o2 ?o2))
   (when (and (Knot_pre_ontable_v1_stack)(Keff_ontable_v1_stack)) (ontable ?o1))
   (when (and (Knot_pre_ontable_v2_stack)(Keff_ontable_v2_stack)) (ontable ?o2))
   (when (and (Knot_pre_clear_v1_stack)(Keff_clear_v1_stack)) (clear ?o1))
   (when (and (Knot_pre_clear_v2_stack)(Keff_clear_v2_stack)) (clear ?o2))
   (when (and (Knot_pre_holding_v1_stack)(Keff_holding_v1_stack)) (holding ?o1))
   (when (and (Knot_pre_holding_v2_stack)(Keff_holding_v2_stack)) (holding ?o2))
   (when (and (Knot_pre_handempty_stack)(Keff_handempty_stack)) (handempty))))
```

Figure 2: PDDL encoding of the editable version of the `stack(?v1,?v2)` schema.

References

[Bonet and Geffner, 2000] Blai Bonet and Hector Geffner. Planning with incomplete information as heuristic search in belief space. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems*, pages 52–61. AAAI Press, 2000.

[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.

[Goldman and Boddy, 1996] Robert P Goldman and Mark S Boddy. Expressive planning and explicit knowledge. In *AIPS*, volume 96, pages 110–117, 1996.

[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 Artifical Intelligence, (IJCAI-09)*, pages 1778–1783. AAAI Press, 2009.

[Smith and Weld, 1998] David E Smith and Daniel S Weld. Conformant graphplan. In *AAAI/IAAI*, pages 889–896, 1998.

[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.

[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.