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 a sequence of partially observed states, this paper formalizes the task of computing the exact set of action models that are *conformant* with the given observation. This task is of interest because it allows to, incrementally and scalably, learn action models from arbitrary large sets of state observations.

In addition, the paper presets a new method to compute *the least commitment* action model from state observations. The method assumes that action models are specified as STRIPS action schema and builds on top of off-the-shelf *conformant planning* algorithms.

2 Background

This section formalizes the *classical* and *conformant* planning models as well as the kind of input observations for the computation of *the least commitment* action model.

2.1 Classical planning with conditional effects

F is the set of *fluents* or *state variables* (propositional variables). A *literal* l is a valuation of a fluent $f \in F$, i.e. either $l = f$ or $l = \neg f$. L is a set of literals that represents a partial assignment of values to fluents, and $\mathcal{L}(F)$ is the set of all literals sets on F, i.e. all partial assignments of values to fluents. A *state* s is a full assignment of values to fluents. We explicitly include negative literals $\neg f$ in states s.t. $|s| = |F|$ and the size of the state space is $2^{|F|}$.

A *planning frame* is a tuple $\Phi = \langle F, A \rangle$, where F is a set of fluents and A is a set of *actions*. An action $a \in A$ is defined with *preconditions*, $\mathsf{pre}(a) \in \mathcal{L}(F)$, *positive effects*, $\mathsf{eff}^+(a) \in \mathcal{L}(F)$, and *negative 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 $\mathsf{pre}(a) \subseteq s$. And the result of applying a in s is $\theta(s, a) = \{s \setminus \mathsf{eff}^-(a)) \cup \mathsf{eff}^+(a)\}$.

A *classical planning problem* is a tuple $P = \langle F, A, I, G \rangle$, where I is the initial state in which all the fluents of F are assigned a *true*|*false* value and $G \in \mathcal{L}(F)$ is the set of goal conditions over the state variables. A *plan* π for P is an action sequence $\pi = \langle a_1, \ldots, a_n \rangle$, and $|\pi| = n$ denotes its *plan length*. The execution of π in 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 $G \subseteq s_n$.

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 *triggered effects* resulting from the action application are the effects whose conditions hold in s:

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

The result of applying action a_c in state s is $\theta(s, a_c) = \{s \setminus \mathsf{eff}_c^-(s, a)) \cup \mathsf{eff}_c^+(s, a)\}$, where $\mathsf{eff}_c^-(s, a) \subseteq triggered(s, a)$ and $\mathsf{eff}_c^+(s, a) \subseteq triggered(s, a)$ are, respectively, the triggered *negative* and *positive* effects.

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 sequence of partial states that represents the observation from the execution of π in P. Formally, $\mathcal{O}(\tau) = \langle s_0^o, s_1^o \ldots, s_m^o \rangle$ where $s_0^o = I$.

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 $\tau(\pi, s_0)$, meaning that $\forall i, s_i^o \subseteq s_i$. In practice, the number of observed states, m, range 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 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 $(\theta(s_i^o, \langle a_1, \ldots, a_k \rangle) = s_{i+1}^o$, where $k \geq 1$ is unknown but finite. In other words, having $\mathcal{O}(\tau)$ does not imply knowing the actual length of π.

2.3 Conformant planning

Conformant planning refers to planning with incomplete information about the initial state, no sensing, and where goals have to be achieved with certainty (despite the uncertainty of the initial state) [Smith and Weld, 1998; Goldman and Boddy, 1996].

Syntactically, conformant planning problems are expressed in compact form through a set of state variables. A *conformant planning problem* is then 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 the classical planing model). 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* $\tau(\pi, s_0) = \langle s_0, a_1, s_1, \ldots, a_n, s_n \rangle$, such that s_0 is a valuation of the fluents in F that satisfies Υ, then the $\tau(\pi, s_0)$ trajectory reaches a final state $G \subseteq s_n$.

3 Learning *the least commitment* action model from state observations

Now we are ready to formalize the learning of the *the least commitment* action model from state observations and our approach to address it via *conformant planning*.

3.1 The *least commitment* action model

The task of computing *the least commitment* action model is defined as a tuple $\langle P, M, \mathcal{O} \rangle$ where:

- $P = \langle F, A[\cdot], I, G[\cdot] \rangle$ is a planning problem where $A[\cdot]$ is a set of actions. For each $a \in A[\cdot]$, the semantics of a is unknown; i.e. the functions ρ and/or θ of a are undefined. The set of goal conditions $G[\cdot]$ is also unknown.
- M is the set of different planning models for the actions in $A[\cdot]$. A model $\mathcal{M} \in M$ defines the semantics of every action in $A[\cdot]$. Planning models differ in the $\langle \rho, \theta \rangle$ functions of the actions but they all use the same set of state variables F.
- $\mathcal{O}(\tau)$ is a sequence of state observations coming from the observation of a trajectory $\tau(\pi, s_0)$ produced by the execution of certain unknown plan π.

The *solution* to this task is *the least commitment* action model which defines the smallest subset of models $M^* \subseteq M$ that is *conformant* with the input observation $\mathcal{O}(\tau)$.

3.2 The space of possible action models

This work focuses on the lerning of action models that are specified as STRIPS action schema. STRIPS action schemata provide a compact representation for specifying action models. *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.

Definition 1 (Comparable STRIPS action schemata) *Two* STRIPS *schemata ξ and ξ' are* **comparable** *iff $pars(\xi) = pars(\xi')$, i.e, both share the same list of parameters.*[1]

For instance, the `stack(?v1,?v2)` and `unstack(?v1,?v2)` schemata from a four operator *blocksworld* [Slaney and Thiébaux, 2001] are *comparable* while `stack(?v1,?v2)` and `pickup(?v1)` are not. Last but not least, we say that two STRIPS models $\mathcal{M}$ and $\mathcal{M}'$ are *comparable* iff there exists a bijective function that maps every action schema $\xi \in \mathcal{M}$ to a comparable schemata $\xi' \in \mathcal{M}'$ and vice versa.

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 $\mathcal{I}_{\Psi,\xi}$.

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 constrained by a set $\mathcal{C}$ that includes:

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 `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 ξ because of the semantic of the `on` predicate (i.e. only one block can be on top of another block). Invariants constraining the space of possible states in a given domain belong also to this group [Fox and Long, 1998].

Definition 2 (Well-defined STRIPS action schemata) *Given a set of* predicates *Ψ, a list of action* parameters *$pars(\xi)$, and set of FOL constraints $\mathcal{C}$, ξ is a* **well-defined STRIPS action schema** *iff its three lists $pre(\xi) \subseteq \mathcal{I}_{\Psi,\xi}$, $del(\xi) \subseteq \mathcal{I}_{\Psi,\xi}$ and $add(\xi) \subseteq \mathcal{I}_{\Psi,\xi}$ only contain elements in $\mathcal{I}_{\Psi,\xi}$ and they satisfy all the constraints in $\mathcal{C}$.*

We say a planning model $\mathcal{M}$ is *well-defined* if all its STRIPS action schemata are *well-defined*.

3. *Observation constraints*. A sequence of state observations $\mathcal{O}(\tau)$ constraints further the space of possible action schemata. This *semantic knowledge* included in the observations introduce a third type of constraints and can also be added to the set $\mathcal{C}$.

[1] In STRIPS models, $pars(\xi) = pars(\xi')$ implies the number of parameters must be the same. For other planning models that allow object typing, the equality implies that parameters share the same type

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


(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 action.

3.3 Learning *the least commitment* model with conformant planning

Given a classical planning problem $P = \langle F, A[\cdot], I, G[\cdot] \rangle$ and a sequence of state observations $\mathcal{O}(\tau)$ coming from the observation of a $\tau(\pi, I)$ trajectory, we show here that we can build a *conformant planning problem* P_c in linear time and space and whose solution is a plan that induces the *least commitment* model that is conformant with $\mathcal{O}(\tau)$.

In more detail, our compilation defines a *conformant planning problem* $P_c = \langle F_c, A_c, \Upsilon, G \rangle$ such that:

- The set of fluents F_c 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 fluents $[pre|del|add]\_e\_\xi$ such that $e \in \mathcal{I}_{\Psi,\xi}$ for a propositional encoding of the *preconditions*, *negative* and *positive* effects of schema ξ. Figure 1 shows the propositional encoding of the unstack action from the blocksworld.
- The set of actions A_c contains now actions of three different kinds:
  - Actions for *committing* pre_e_ξ fluents to a positive and a negative value.

$$\begin{aligned}
\mathsf{pre}(\mathsf{set}\top_\mathsf{pre_e_}\xi) &= \{\},\\
\mathsf{cond}(\mathsf{set}\top_\mathsf{pre_e_}\xi) &= \{pre_e_\xi\} \rhd \{pre_e_\xi\},\\
&\quad \{\neg pre_e_\xi\} \rhd \{pre_e_\xi\}.\\
\mathsf{pre}(\mathsf{set}\bot_\mathsf{pre_e_}\xi) &= \{\},\\
\mathsf{cond}(\mathsf{set}\bot_\mathsf{pre_e_}\xi) &= \{pre_e_\xi\} \rhd \{\neg pre_e_\xi\},\\
&\quad \{\neg pre_e_\xi\} \rhd \{\neg pre_e_\xi\}.
\end{aligned}$$

    In addition, similar actions are defined for *committing* eff_e_ξ fluents to a positive value and for *committing* pre_e_ξ fluents to a negative value.
  - Actions for *validating* the edited models at the s_j observed states, $0 \leq j < m$

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

  - Actions whose semantics is given by the value of the pre_e_ξ, pre_e_ξ fluents at the current state.

```
(:action stack
 :parameters (?o1 - object ?o2 - object)
 :precondition
   (and (or (not (pre_on_v1_v1_stack)) (on ?o1 ?o1))
        (or (not (pre_on_v1_v2_stack)) (on ?o1 ?o2))
        (or (not (pre_on_v2_v1_stack)) (on ?o2 ?o1))
        (or (not (pre_on_v2_v2_stack)) (on ?o2 ?o2))
        (or (not (pre_ontable_v1_stack)) (ontable ?o1))
        (or (not (pre_ontable_v2_stack)) (ontable ?o2))
        (or (not (pre_clear_v1_stack)) (clear ?o1))
        (or (not (pre_clear_v2_stack)) (clear ?o2))
        (or (not (pre_holding_v1_stack)) (holding ?o1))
        (or (not (pre_holding_v2_stack)) (holding ?o2))
        (or (not (pre_handempty_stack)) (handempty)))
 :effect (and
   (when (and (pre_on_v1_v1_stack)(eff_on_v1_v1_stack)) (not (on ?o1 ?o1)))
   (when (and (pre_on_v1_v2_stack)(eff_on_v1_v2_stack)) (not (on ?o1 ?o2)))
   (when (and (pre_on_v2_v1_stack)(eff_on_v2_v1_stack)) (not (on ?o2 ?o1)))
   (when (and (pre_on_v2_v2_stack)(eff_on_v2_v2_stack)) (not (on ?o2 ?o2)))
   (when (and (pre_ontable_v1_stack)(eff_ontable_v1_stack)) (not (ontable ?o1)))
   (when (and (pre_ontable_v2_stack)(eff_ontable_v2_stack)) (not (ontable ?o2)))
   (when (and (pre_clear_v1_stack)(eff_clear_v1_stack)) (not (clear ?o1)))
   (when (and (pre_clear_v2_stack)(eff_clear_v2_stack)) (not (clear ?o2)))
   (when (and (pre_holding_v1_stack)(eff_holding_v1_stack)) (not (holding ?o1)))
   (when (and (pre_holding_v2_stack)(eff_holding_v2_stack)) (not (holding ?o2)))
   (when (and (pre_handempty_stack)(eff_handempty_stack)) (not (handempty)))
   (when (and (not(pre_on_v1_v1_stack))(eff_on_v1_v1_stack)) (on ?o1 ?o1))
   (when (and (not(pre_on_v1_v2_stack))(eff_on_v1_v2_stack)) (on ?o1 ?o2))
   (when (and (not(pre_on_v2_v1_stack))(eff_on_v2_v1_stack)) (on ?o2 ?o1))
   (when (and (not(pre_on_v2_v2_stack))(eff_on_v2_v2_stack)) (on ?o2 ?o2))
   (when (and (not(pre_ontable_v1_stack))(eff_ontable_v1_stack)) (ontable ?o1))
   (when (and (not(pre_ontable_v2_stack))(eff_ontable_v2_stack)) (ontable ?o2))
   (when (and (not(pre_clear_v1_stack))(eff_clear_v1_stack)) (clear ?o1))
   (when (and (not(pre_clear_v2_stack))(eff_clear_v2_stack)) (clear ?o2))
   (when (and (not(pre_holding_v1_stack))(eff_holding_v1_stack)) (holding ?o1))
   (when (and (not(pre_holding_v2_stack))(eff_holding_v2_stack)) (holding ?o2))
   (when (and (not(pre_handempty_stack))(eff_handempty_stack)) (handempty))))
```

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

  - Actions whose semantics is given by the value of the pre_e_ξ, pre_e_ξ fluents at the current state. Given an operator schema $\xi \in \mathcal{M}$ its *editable* version is formalized as:

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

    Figure 2 shows the PDDL encoding of the editable version of the `stack(?v1,?v2)` schema. Note that this editable schema, when the fluents `(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)` hold, it behaves exactly as defined in Figure 1.
- The new goals are $G_c = \{test_m\}$.

Given a plan π that solves the *conformant planning* problem P_c that results from our compilation then, the *least commitment* model that is conformant with $\mathcal{O}(\tau)$ is extracted in linear time from the last state reached by π.

3.4 Compilation properties

4 Evaluation

5 Conclusions

References

[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.

[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.

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

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