Lecture 24

CS 6260 Automated Planning and Learning

FAMA: Model Learning as Planning

Department of Computer Science and Engineering
Indian Institute of Technology Madras

Logistics

Logistics

- 3 Exams:
  1. [Quiz 1] Feb 19: 08:00 AM – 08:50 AM
  2. [Quiz 2] Apr 10: 05:00 PM – 05:50 PM
  3. [End Sem] May 12

- Office Hours: 1 hour after class every week Tu, W, Th
  - No office hours this Wednesday

Recap: Learning Action Models

Learning Action Models from Passive Observations

$\langle s_0, a_1, s_1 \rangle$
$\langle s_1, a_2, s_2 \rangle$
$\vdots$
$\langle s_{n-1}, a_n, s_n \rangle$

[Input]

Learn
er

```
(:action pick-up-beaker
   :parameters (?x - beaker)
    :precondition (and (ontable ?x)
       (not-in-use ?x)
       (handempty))
    :effect (and (not (ontable ?x))
        (not (handempty))
        (holding ?x)))

(:action put-on-shelf
   :parameters (?x - beaker)
    :precondition (holding ?x)
    :effect (and (not (holding ?x))
        (handempty)
        (onshelf ?x)))
```

[PDDL Example]

What kind of approaches these learners use?

Planning based Learners

- Create Planning problem using SAT-like rules.
- Extract correct PDDL from solution to the planning problem.

$\langle s_0, a_1, s_1 \rangle$
$\langle s_1, a_2, s_2 \rangle$
$\vdots$
$\langle s_{n-1}, a_n, s_n \rangle$

[Input]

```
(:action apply_stack
  :parameters (?o1 - object ?o2 - object)
  :precondition
   (and (or (not (pre_stack_on_v1_v1)) (on ?o1 ?o1))
        (or (not (pre_stack_on_v1_v2)) (on ?o1 ?o2))
        (or (not (pre_stack_on_v2_v1)) (on ?o2 ?o1))
        (or (not (pre_stack_on_v2_v2)) (on ?o2 ?o2))
                        . . .
        (or (not (pre_stack_handempty)) (handempty)))
  :effect
   (and (when (del_stack_on_v1_v1) (not (on ?o1 ?o1)))
        (when (del_stack_on_v1_v2) (not (on ?o1 ?o2)))
        (when (del_stack_on_v2_v1) (not (on ?o2 ?o1)))
        (when (del_stack_on_v2_v2) (not (on ?o2 ?o2)))
                        . . .
        (when (add_stack_holding_v1) (holding ?o1))
        (when (add_stack_holding_v2) (holding ?o2))
        (when (add_stack_handempty) (handempty))
        (when (modeProg) (not (modeProg)))))
```

```
(:action pick-up-beaker
   :parameters (?x - beaker)
    :precondition (and (ontable ?x)
       (not-in-use ?x)
       (handempty))
    :effect (and (not (ontable ?x))
        (not (handempty))
        (holding ?x)))

(:action put-on-shelf
   :parameters (?x - beaker)
    :precondition (holding ?x)
    :effect (and (not (holding ?x))
        (handempty)
        (onshelf ?x)))
```

[PDDL Example]

FAMA – Aineto et al. (ICAPS'18, AIJ'19)

FAMA

Notations

- F: Set of fluents.
- l: literal l is a valuation of a fluent $f \in F \implies l = f$ or $l = \neg f$.
- L: Set of literals representing a *partial* assignment of values to fluents.
- $\mathcal{L}(F)$: Set of all literal sets on F.
- s: State, which is full assignment of values to fluents. $|s| = |F|$.

Classical Planning

- 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.
- An action $a \in A$ is defined with: $pre(a) \subseteq \mathcal{L}(F)$, $eff^+(a) \subseteq \mathcal{L}(F)$, and $eff^-(a) \subseteq \mathcal{L}(F)$.
- $a \in A$ is applicable in a state s iff $pre(a) \subseteq s$.
- Result of applying a in s is $\theta(s, a) = \{(s \setminus eff^-(a)) \cup eff^+(a)\}$

Actions with Conditional Effects

An action $a \in A$ with conditional effects is defined as a set of *preconditions* $pre(a)$ and a set of *conditional effects* $cond(a)$.

Each conditional effect $C \triangleright E \in cond(a)$ is composed of two set of literals:

- Condition: $C \subseteq \mathcal{L}(F)$
- Effect: $E \subseteq \mathcal{L}(F)$

Actions with Conditional Effects

Triggered Effects of action a are "effects whose conditions hold in s":

$$triggered(s, a) = \bigcup_{C \triangleright E \in cond(a), C \subseteq s} E$$

Result of applying a in s is $\theta(s, a) = \{(s \setminus eff^-(a)) \cup eff^+(a)\}$, where

- $eff^-(a)) \subseteq triggered(s, a)$ are triggered negative effects of a, and
- $eff^+(a)) \subseteq triggered(s, a)$ are triggered positive effects of a.

Classical Planning Problem

A classical planning problem is a tuple $P = \langle F, A, I, G \rangle$, where:

- I is the initial state, and
- $G \subseteq \mathcal{L}(F)$ is a goal condition.

Plan

A plan for P is an action sequence $\pi = \langle a_1, \ldots, a_n \rangle$ that induces state trajectory $\langle s_0, s_1, \ldots, s_n \rangle$, such that $s_0 = I$ and $a_i (1 \leq i \leq n)$ is applicable in s_{i-1} and generates the successor state $s_i = \theta(s_{i-1}, a_i)$

A plan π solves P iff $G \subseteq s_n$.

Relational Notations

- Ψ: Set of *predicates*.
- $ar(p)$: Arity of argument list of each predicate $p \in \Psi$.
- Ω: Set of objects.
- Ω_v: Variable names. $\Omega_v = \{v_i\}_i^{max_{a \in A} ar(a)}$ and $(\Omega \cap \Omega_v = \emptyset)$

Blocks-world Example

ACTIONS

```
(:action pick-up
          :parameters (?v1)
          :precondition ()
          :effect ())
(:action put-down
          :parameters (?v1)
          :precondition ()
          :effect ())
(:action stack
          :parameters (?v1 ?v2)
          :precondition ()
          :effect ())
(:action unstack
          :parameters (?v1 ?v2)
          :precondition ()
          :effect ())
```

1 parameter

1 parameter

2 parameters

$|\Omega_v| = 2$

2 parameters

PREDICATES

```
(:predicates (on ?v1 ?v2)
             (ontable ?v1)
             (clear ?v1)
             (handempty)
             (holding ?v1))
```

PROBLEM

```
(define (problem BLOCKS-7-1)
(:domain BLOCKS)
(:objects E B D F G C A )
(:INIT (CLEAR A) (CLEAR C) (ONTABLE G)
       (ONTABLE F) (ON A G) (ON C D) (ON D B)
       (ON B E) (ON E F) (HANDEMPTY))
(:goal (AND (ON A E) (ON E B) (ON B F)
       (ON F G) (ON G C) (ON C D)))
)
```

Learning Action Models

Input: Initial Domain Model $\mathcal{M}$ and Plan Traces

Plan Trace $\tau = \langle s_0, a_1, s_1, a_2, s_2, \ldots, a_n, s_n \rangle$
Each plan trace has two components:

- State Trajectory: $\tau_s = \langle s_0, s_1, \ldots s_n \rangle$
- Action Sequence: $\tau_a = \langle a_0, a_1, \ldots a_n \rangle$

Output: Learned Domain Model $\mathcal{M}'$

Domain model $\mathcal{M}'$ is consistent with the information of $\mathcal{M}$ and with the observed plan traces.

Observability of State Trajectories

Fully Observable (FO) State Trajectories

Every observed intermediate state of plan trace is a full assignment of values to fluents, and there exists a single action for every transition.
$\forall i, 1 \leq i < n, \forall s_i \in \tau_s, |s_i| = |F| \wedge \theta(s_i, \langle a \rangle) = s_{i+1}$

Partially Observable (PO) State trajectories

At least one intermediate state of τ_s is a partial assignment of values to fluents.
$\exists i, 1 \leq i < n, |s_i| < |F|$

Partiall Observable (PO) State trajectories

There are 2 special cases:

Non-Observable (NO) State trajectories

When all of the $n-1$ intermediate states of s are missing in τ_s.
$\forall i, 1 \leq i < n, s_i = \emptyset; i.e., |s_i| = 0$

PO* State trajectories

When none of the $n-1$ intermediate states of s are missing in τ_s.
$\exists i, 1 \leq i < n, 0 < |s_i| < |F|$ and $\forall i, 1 \leq i < n, s_i \neq \emptyset$

Observability of Action Sequences

Fully Observable (FO) Action Sequences

All of the actions of π appear in τ_a, i.e., $\pi = \tau_a$.
$\forall i, 1 \leq i < n, a_i \in \pi \wedge a_i \in \tau_a$

Partially Observable (PO) Action Sequences

At least one of the necessary actions of the plan π is missing in τ_a.
$\exists i, 1 \leq i < n, a_i \in \pi \wedge a_i \notin \tau_a$

Non Observable (NO) Action Sequences

None of the actions of π appear in τ_a, i.e., $\tau_a = \emptyset$
$\forall i, 1 \leq i < n, a_i \in \pi \wedge a_i \notin \tau_a$

Learning Task

The learning task is formalized by the pair $\Lambda = \langle \mathcal{M}, \tau \rangle$, where:

- $\mathcal{M}$ is the initial domain model. $\mathcal{M} = \emptyset$ when learning from scratch, or $\mathcal{M} = \Xi_0$ when learning from a partially known model.
- τ is the observed plan trace such that:
  - Observations in τ are noiseless.
  - $s_0 \in \tau$ is a fully observed state including positive and negative fluents; i.e., $|s_0| = |F|$.
  - The header of an action model is either given by $\mathcal{M}$ or inferable from τ.

Assume that Λ will contain the predicates Ψ as well as the headers of the actions models, either explicitly provided in $\mathcal{M}$ or deducible from τ.

Sample Input

```
;;;;;; Action headers in M

(pickup ?v1) (unstack ?v1 ?v2)


;;;;;; Plan trace τ

;;; Initial state observation
(clear B) (ontable A) (handempty) (on B A)
(not (clear A)) (not (ontable B)) (not (holding A)) (not (holding B))
(not (on A A)) (not (on A B)) (not (on B B))

;;; Action observation
(putdown B)

;;; Action observation
(stack A B)

;;; Final state observation
(clear A) (on A B) (ontable B)
```

Sample Input

```
;;;;;; Action headers in M

(pickup ?v1) (unstack ?v1 ?v2)
```

These 2 have to be mentioned here, because they don't appear in plan trace.

```
;;;;;; Plan trace τ

;;; Initial state observation
(clear B) (ontable A) (handempty) (on B A)
(not (clear A)) (not (ontable B)) (not (holding A)) (not (holding B))
(not (on A A)) (not (on A B)) (not (on B B))
```

Missing action observation: (unstack B A)

```
;;; Action observation
(putdown B)
```

Missing action observation: (pickup A)

```
;;; Action observation
(stack A B)

;;; Final state observation
(clear A) (on A B) (ontable B)
```

Solving Learning Task: Novel Contribution of the Paper

Transform the learning task Λ into a planning task P_Λ.

Solution π_Λ of P_Λ will contain 2 differentiated kinds of actions:

- Actions defining **insertion** of a fluent as precondition, a positive effect, or a negative effect of an action model $\xi \in \mathcal{M}'$.
- Actions determining the application of the learned ξs while successively **validating** the effects of the action application in every observable point of τ.

Solving Learning Task: Novel Contribution of the Paper

Transform the learning task Λ into a planning task P_Λ.

Solution π_Λ of P_Λ will contain 2 differentiated kinds of actions:

- Actions defining **insertion** of a fluent as precondition, a positive effect, or a negative effect of an action model $\xi \in \mathcal{M}'$.
- Actions determining the application of the learned ξs while successively **validating** the effects of the action application in every observable point of τ.

Rest of the presentation will focus on how this transformation from Λ to P_Λ is done.

Propositional Encoding: Instantiating Predicates

Assign objects in Ω to the arguments of the predicates in Ψ, i.e.
$\Psi_v = p(\omega) : p \in \Psi, \omega \in \Omega^{ar(p)}$

Example with $\Omega_v = \{v_1, v_2\}$

$\Psi_v = \{handempty,$

Propositional Encoding: Instantiating Predicates

Assign objects in Ω to the arguments of the predicates in Ψ, i.e.
$\Psi_v = p(\omega) : p \in \Psi, \omega \in \Omega^{ar(p)}$

Example with $\Omega_v = \{v_1, v_2\}$

$\Psi_v = \{$*handempty*,
holding(v_1), *holding*(v_2),

Propositional Encoding: Instantiating Predicates

Assign objects in Ω to the arguments of the predicates in Ψ, i.e.
$\Psi_v = p(\omega) : p \in \Psi, \omega \in \Omega^{ar(p)}$

Example with $\Omega_v = \{v_1, v_2\}$

$\Psi_v = \{$*handempty*,
holding(v_1), *holding*(v_2),
clear(v_1), *clear*(v_2),

Propositional Encoding: Instantiating Predicates

Assign objects in Ω to the arguments of the predicates in Ψ, i.e.
$\Psi_v = p(\omega) : p \in \Psi, \omega \in \Omega^{ar(p)}$

Example with $\Omega_v = \{v_1, v_2\}$

$\Psi_v = \{$*handempty*,
holding(v_1), *holding*(v_2),
clear(v_1), *clear*(v_2),
ontable(v_1), *ontable*(v_2),

Propositional Encoding: Instantiating Predicates

Assign objects in Ω to the arguments of the predicates in Ψ, i.e.
$\Psi_v = p(\omega) : p \in \Psi, \omega \in \Omega^{ar(p)}$

Example with $\Omega_v = \{v_1, v_2\}$

$$\begin{aligned}\Psi_v = \{&\mathit{handempty},\\ &\mathit{holding}(v_1),\ \mathit{holding}(v_2),\\ &\mathit{clear}(v_1),\ \mathit{clear}(v_2),\\ &\mathit{ontable}(v_1),\ \mathit{ontable}(v_2),\\ &\mathit{on}(v_1, v_1),\ \mathit{on}(v_1, v_2),\ \mathit{on}(v_2, v_1),\ \mathit{on}(v_2, v_2)\}\end{aligned}$$

Propositional Encoding: Instantiating Actions

An action $a \in A$ is instantiated from a STRIPS operator schema $\xi = \langle head(\xi), pre(\xi), add(\xi), del(\xi) \rangle$ where:

- $head(\xi) = \langle name(\xi), pars(\xi) \rangle$: Operator header where *name* is the *actionname*, and $pars(\xi) = \{v_i\}_{i=1}^{ar(\xi)}$.
  Ex: $header = \{pickup(v_1), putdown(v_1), stack(v_1, v_2), unstack(v_1, v_2)\}$ for the blocksworld example.

- $pre(\xi) \subseteq F_v$, $add(\xi) \subseteq F_v$, and $del(\xi) \subseteq F_v$ are preconditions, positive effects, and negative effects respectively.

Propositional Encoding: Instantiating Actions

For a given action model ξ, we define $\Psi_\xi \subseteq \Psi_v$ as the subset of elements of Ψ_v that can appear in ξ.

Example

$\Psi_{stack} = \Psi_v$

$\Psi_{pickup} = \{handempty, holding(v_1), clear(v_1), ontable(v_1), on(v_1, v_1)\}$

Propositional Encoding: Instantiating Actions

For a given action model ξ, we define $\Psi_\xi \subseteq \Psi_v$ as the subset of elements of Ψ_v that can appear in ξ.

Example

$\Psi_{stack} = \Psi_v$

$\Psi_{pickup} = \{handempty, holding(v_1), clear(v_1), ontable(v_1), on(v_1, v_1)\}$

We now create propositional encoding of the model fluents of $pre(\xi)$, $del(\xi)$ and $add(\xi)$.

Propositional Encoding: Instantiating Actions

For every ξ and $p \in \Psi_\xi$, we create:

- $pre_p(\xi)$: pre + $name(\xi)$ + (fluent of arity 0 using elements of p).
  Example: $pre_p(\xi) =$ `pre_stack_on_v1_v2`, when $name(\xi) = stack$ and $p = on(v_1, v_2)$.

- $del_p(\xi)$: del + $name(\xi)$ + (fluent of arity 0 using elements of p).
  Example: $del_p(\xi) =$ `del_stack_on_v1_v2`, when $name(\xi) = stack$ and $p = on(v_1, v_2)$.

- $add_p(\xi)$: add + $name(\xi)$ + (fluent of arity 0 using elements of p).
  Example: $add_p(\xi) =$ `add_stack_on_v1_v2`, when $name(\xi) = stack$ and $p = on(v_1, v_2)$.

Planning Task

Given a learning task $\Lambda = \langle \mathcal{M}, \tau \rangle$, with τ formed by an n-action sequence $\langle a_1, \ldots, a_n \rangle$ and a m-state trajectory $\langle s_0, s_1, \ldots, s_m \rangle$ ($\tau = \langle s_0, a_1, \ldots, a_n, s_m \rangle$), the compilation outputs a classical planning task $P_\Lambda = \langle F_\Lambda, A_\Lambda, I_\Lambda, G_\Lambda \rangle$.

Planning Task

Given a learning task $\Lambda = \langle \mathcal{M}, \tau \rangle$, with τ formed by an n-action sequence $\langle a_1, \ldots, a_n \rangle$ and a m-state trajectory $\langle s_0, s_1, \ldots, s_m \rangle$ ($\tau = \langle s_0, a_1, \ldots, a_n, s_m \rangle$), the compilation outputs a classical planning task $P_\Lambda = \langle F_\Lambda, A_\Lambda, I_\Lambda, G_\Lambda \rangle$.

Lets see one by one how we get $F_\Lambda, A_\Lambda, I_\Lambda$, and G_Λ.

F_Λ

In addition to F, F_Λ contains:

- Model fluents $pre_p(\xi)$, $del_p(\xi)$, and $add_p(\xi)$, for every $\xi \in \mathcal{M}$ and $p \in \Psi_\xi$.

F_Λ

In addition to F, F_Λ contains:

- Model fluents $pre_p(\xi)$, $del_p(\xi)$, and $add_p(\xi)$, for every $\xi \in \mathcal{M}$ and $p \in \Psi_\xi$.
- A set of fluents $F_\pi = \{plan(name(a_i), \Omega^{ar(a_i)}, i)\}_{1 \leq i \leq n}$ to represent i^{th} observable action of τ.

F_Λ

In addition to F, F_Λ contains:

- Model fluents $pre_p(\xi)$, $del_p(\xi)$, and $add_p(\xi)$, for every $\xi \in \mathcal{M}$ and $p \in \Psi_\xi$.
- A set of fluents $F_\pi = \{plan(name(a_i), \Omega^{ar(a_i)}, i)\}_{1 \leq i \leq n}$ to represent i^{th} observable action of τ.

Example

```
;;; Action observation
(putdown B)

;;; Action observation
(stack A B)
```

This will be encoded as:

`(plan-putdown B i1)`, `(plan-stack A B i2)`

F_Λ

In addition to F, F_Λ contains:

- Model fluents $pre_p(\xi)$, $del_p(\xi)$, and $add_p(\xi)$, for every $\xi \in \mathcal{M}$ and $p \in \Psi_\xi$.
- A set of fluents $F_\pi = \{plan(name(a_i), \Omega^{ar(a_i)}, i)\}_{1 \leq i \leq n}$ to represent i^{th} observable action of τ.
- Two fluents to iterate through the n observed actions of τ. Given $1 \leq i \leq n$:
  - at_i: Ensures that actions are executed in the same order as in τ.
  - $next_{i,i+1}$: iterates to the next planning step when solving P_Λ.

F_Λ

In addition to F, F_Λ contains:

- Model fluents $pre_p(\xi)$, $del_p(\xi)$, and $add_p(\xi)$, for every $\xi \in \mathcal{M}$ and $p \in \Psi_\xi$.
- A set of fluents $F_\pi = \{plan(name(a_i), \Omega^{ar(a_i)}, i)\}_{1 \leq i \leq n}$ to represent i^{th} observable action of τ.
- Two fluents to iterate through the n observed actions of τ. Given $1 \leq i \leq n$:
  - at_i: Ensures that actions are executed in the same order as in τ.
  - $next_{i,i+1}$: iterates to the next planning step when solving P_Λ.
- Set of fluents $\{test_j\}_{0 \leq j \leq m}$, to point at the state observation $s_j \in \tau$ where the action model is validated.

F_Λ

In addition to F, F_Λ contains:

- Model fluents $pre_p(\xi)$, $del_p(\xi)$, and $add_p(\xi)$, for every $\xi \in \mathcal{M}$ and $p \in \Psi_\xi$.
- A set of fluents $F_\pi = \{plan(name(a_i), \Omega^{ar(a_i)}, i)\}_{1 \leq i \leq n}$ to represent i^{th} observable action of τ.
- Two fluents to iterate through the n observed actions of τ. Given $1 \leq i \leq n$:
  - at_i: Ensures that actions are executed in the same order as in τ.
  - $next_{i,i+1}$: iterates to the next planning step when solving P_Λ.
- Set of fluents $\{test_j\}_{0 \leq j \leq m}$, to point at the state observation $s_j \in \tau$ where the action model is validated.

Example

For the earlier example where $\tau = \langle s_0, (putdown\ B), (stack\ A\ B), s_4 \rangle$, two tests are required to validate the programmed action model, one test at s_0 and another one at s_4.

F_Λ

In addition to F, F_Λ contains:

- Model fluents $pre_p(\xi)$, $del_p(\xi)$, and $add_p(\xi)$, for every $\xi \in \mathcal{M}$ and $p \in \Psi_\xi$.
- A set of fluents $F_\pi = \{plan(name(a_i), \Omega^{ar(a_i)}, i)\}_{1 \leq i \leq n}$ to represent i^{th} observable action of τ.
- Two fluents to iterate through the n observed actions of τ. Given $1 \leq i \leq n$:
  - at_i: Ensures that actions are executed in the same order as in τ.
  - $next_{i,i+1}$: iterates to the next planning step when solving P_Λ.
- Set of fluents $\{test_j\}_{0 \leq j \leq m}$, to point at the state observation $s_j \in \tau$ where the action model is validated.
- A fluent, $mode_{prog}$, to indicate whether action models are being programmed or validated.

F_Λ

In addition to F, F_Λ contains:

- Model fluents $pre_p(\xi)$, $del_p(\xi)$, and $add_p(\xi)$, for every $\xi \in \mathcal{M}$ and $p \in \Psi_\xi$.
- A set of fluents $F_\pi = \{plan(name(a_i), \Omega^{ar(a_i)}, i)\}_{1 \leq i \leq n}$ to represent i^{th} observable action of τ.
- Two fluents to iterate through the n observed actions of τ. Given $1 \leq i \leq n$:
  - at_i: Ensures that actions are executed in the same order as in τ.
  - $next_{i,i+1}$: iterates to the next planning step when solving P_Λ.
- Set of fluents $\{test_j\}_{0 \leq j \leq m}$, to point at the state observation $s_j \in \tau$ where the action model is validated.
- A fluent, $mode_{prog}$, to indicate whether action models are being programmed or validated.
- A fluent, $action_{applied}$, to force the execution of at least one action between two observed states.

I_Λ and G_Λ

I_Λ contains:

- s_0
- $mode_{prog}$
- $test_0$
- F_π
- at_1
- $\{next_{i,i+1}\}$ where $1 \leq i \leq n$.

G_Λ contains:

- $test_m$
- at_n

A_Λ

A_Λ includes three types of actions:

- Actions for *inserting* a component (precondition, positive effect or negative effect) in $\xi \in \mathcal{M}$ following the syntactic constraints of STRIPS models. These can be further subdivided into 2 categories:
  - Actions which support the addition of a precondition.
  - Actions which support the addition of a positive or negative effect.
- Actions for *applying* the action models $\xi \in \mathcal{M}$ built by the insert actions and bounded to objects $\omega \in \Omega^{ar(\xi)}$.
- Actions for *validating* the partially observed state $s_j \in \tau, 1 \leq j < m$.

A_Λ: Insert Actions

Inserting Precondition

Actions which support the addition of a precondition $p \in \Psi_\xi$ to the action model $\xi \in \mathcal{M}$.

$$pre(insertPre_{p,\xi}) = \{\neg pre_p(\xi), \neg del_p(\xi), \neg add_p(\xi), mode_{prog}\},$$
$$cond(insertPre_{p,\xi}) = \{\emptyset\} \triangleright \{pre_p(\xi)\}.$$

Inserting Effects

Actions which support the addition of a negative or positive effect $p \in \Psi_\xi$ to the action model $\xi \in \mathcal{M}$.

$$\begin{aligned} pre(insertEff_{p,\xi}) =& \{\neg del_p(\xi), \neg add_p(\xi), mode_{prog}\}, \\ cond(insertEff_{p,\xi}) =& \{pre_p(\xi)\} \triangleright \{del_p(\xi)\}, \\ & \{\neg pre_p(\xi)\} \triangleright \{add_p(\xi)\}. \end{aligned}$$

A_Λ: Insert Actions Example

Given $name(\xi) = stack$ and $C_{pre\_stack} = \{$ (pre_stack_holding_v1), (pre_stack_holding_v2), (pre_stack_on_v1_v2), (pre_stack_clear_v1),... $\}$.

Example

```
(:action insert_pre_stack_on_v1_v1
    :parameters ()
    :precondition (and (modeProg)(not (pre_stack_on_v1_v1))
            (not (del_stack_on_v1_v1))(not (add_stack_on_v1_v1)))
    :effect (and (pre_stack_on_v1_v1)))

(:action insert_eff_stack_on_v1_v1
    :parameters ()
    :precondition (and (modeProg)
            (not (del_stack_on_v1_v1))(not (add_stack_on_v1_v1)))
    :effect (and (when (pre_stack_on_v1_v1)(del_stack_on_v1_v1))
            (when (not (pre_stack_on_v1_v1))(add_stack_on_v1_v1))))
```

A_Λ: Apply Actions; NO case

Since action headers are known, the variables $pars(\xi)$ are bounded to the objects in ω that appear in the same position.

$$
\begin{aligned}
pre(apply_{\xi,\omega}) =& \neg action_{applied} \cup \{pre_p(\xi) \implies p(\omega)\}_{\forall p \in \Psi_\xi}, \\
cond(apply_{\xi,\omega}) =& \{del_p(\xi)\} \triangleright \{\neg p(\omega)\}_{\forall p \in \Psi_\xi}, \\
& \{add_p(\xi)\} \triangleright \{p(\omega)\}_{\forall p \in \Psi_\xi}, \\
& \{\emptyset\} \triangleright \{action_{applied}\}, \\
& \{mode_{prog}\} \triangleright \{\neg mode_{prog}\}.
\end{aligned}
$$

A_Λ: Apply Actions Example

```
  (:action apply_stack
:parameters (?o1 - object ?o2 - object)
:precondition
(and (or (not(pre_stack_on_v1_v1)) (on ?o1 ?o1))
(or (not(pre_stack_on_v1_v2)) (on ?o1 ?o2))
(or (not(pre_stack_on_v2_v1)) (on ?o2 ?o1))
(or (not(pre_stack_on_v2_v2)) (on ?o2 ?o2))
(or (not(pre_stack_ontable_v1)) (ontable ?o1))
(or (not(pre_stack_ontable_v2)) (ontable ?o2))
(or (not(pre_stack_clear_v1)) (clear ?o1))
(or (not(pre_stack_clear_v2)) (clear ?o2))
(or (not(pre_stack_holding_v1)) (holding ?o1))
(or (not(pre_stack_holding_v2)) (holding ?o2))
(or (not(pre_stack_handempty)) (handempty))
(not(action_applied)))
:effect
(and (when (del_stack_on_v1_v1) (not(on ?o1 ?o1)))
(when (del_stack_on_v1_v2) (not(on ?o1 ?o2)))
(when (del_stack_on_v2_v1) (not(on ?o2 ?o1)))
(when (del_stack_on_v2_v2) (not(on ?o2 ?o2)))
(when (del_stack_ontable_v1) (not(ontable ?o1)))
  (when (del_stack_ontable_v2) (not(ontable ?o2)))
(when (del_stack_clear_v1) (not(clear ?o1)))
(when (del_stack_clear_v2) (not(clear ?o2)))
(when (del_stack_holding_v1) (not(holding ?o1)))
(when (del_stack_holding_v2) (not(holding ?o2)))
(when (del_stack_handempty) (not(handempty)))
(when (add_stack_on_v1_v1) (on ?o1 ?o1))
(when (add_stack_on_v1_v2) (on ?o1 ?o2))
(when (add_stack_on_v2_v1) (on ?o2 ?o1))
(when (add_stack_on_v2_v2) (on ?o2 ?o2))
(when (add_stack_ontable_v1) (ontable ?o1))
(when (add_stack_ontable_v2) (ontable ?o2))
(when (add_stack_clear_v1) (clear ?o1))
(when (add_stack_clear_v2) (clear ?o2))
(when (add_stack_holding_v1) (holding ?o1))
(when (add_stack_holding_v2) (holding ?o2))
(when (add_stack_handempty) (handempty))
(when (modeProg) (not(modeProg)))
(action_applied)))
```

A_Λ: Apply Actions; PO and FO case

PO Case

If the input plan trace contains PO actions, $cond(apply_{\xi,\omega})$ also includes: $\{at_i, plan(name(a_i), \Omega^{ar(a_i)}, i)\} \triangleright \{\neg at_i, at_{i+1}\}_{\forall i \in [1,n]}$

FO Case

If the input plan trace contains FO actions,
$pre(apply_{\xi,\omega})$ also includes $\{at_i, plan(name(a_i), \Omega^{ar(a_i)}, i)\}$, and
$cond(apply_{\xi,\omega})$ includes $\{\emptyset\} \triangleright \{\neg at_i, at_{i+1}\}_{\forall i \in [1,n]}$.

A_Λ: Validate Actions

Aimed at checking that the observable data of the input plan trace τ follows after the execution of the apply actions.

$$pre(validate_j) = s_j \cup \{test_{j-1}\},$$
$$eff(validate_j) = \{\emptyset\} \triangleright \{\neg test_{j-1}, test_j\}.$$

- There will be a validate action in π_A for every observed state in τ.
- FOr PO and FO actions, an extra precondition, at_{i+1} is added, where i is the index of the last observed action before the state we are validating.

A_Λ: Validate Actions Example

```
(:action validate_2
    :parameters ()
    :precondition (and (not (modeProg ))(clear a)(clear b)(clear c)(clear d)
        (clear e)(not (clear f))(not (clear g)) (handempty) (not (holding a))
        (not (holding b))(not (holding c)) (not (holding d))(not (holding e))
        (not (holding f))(not (holding g))(not (on a a))(not (on a b))
        (not (on a c))(not (on a d))(not (on a e))(not (on a f))(on a g)
        (not(on b a))(not (on b b))(not (on b c))(not (on b d))(not (on b e))
        (not (on b f))(not (on b g))(not (on c a))(not (on c b))(not (on c c))
        (not (on c d))(not (on c e))(not (on c f))(not (on c g))(not (on d a))
        (not (on d b))(not (on d c))(not (on d d))(not (on d e))(not (on d f))
        (not (on d g))(not (on e a))(not (on e b))(not (on e c))(not (on e d))
        (not (on e e))(on e f)(not (on e g))(not (on f a))(not (on f b))
        (not (on f c))(not (on f d))(not (on f e))(not (on f f))(not (on f g))
        (not (on g a))(not (on g b))(not (on g c))(not (on g d))(not (on g e))
        (not (on g f))(not (on g g))(not (ontable a))(ontable b)(ontable c)
        (ontable d)(not (ontable e))(ontable f)(ontable g)(test1))
    :effect (and (test2)(not (test1))))
```

Example Plan

Assume the same example as before, and that action models for pickup, putdown and unstack are already known.

```
;;;;;; Action headers in M
(pickup ?v1) (unstack ?v1 ?v2)
;;;;;; Plan trace
;;; Initial state observation
(clear B) (ontable A) (handempty)
(on B A)(not (clear A))(not (ontable B))
(not (holding A))(not (holding B))
(not (on A A))(not (on A B))(not (on B B))
;;; Action observation
(putdown B)
;;; Action observation
(stack A B)
;;; Final state observation
(clear A) (on A B) (ontable B)
```

Output Plan

```
00 : (insert pre stack holding v1)

01 : (insert pre stack clear v2)
02 : (insert eff stack clear v1)
03 : (insert eff stack clear v2)
04 : (insert eff stack handempty)
05 : (insert eff stack holding v1)
06 : (insert eff stack on v1 v2)
07 : (apply unstack blockB blockA i1 i2)
08 : (apply putdown blockB i2 i3)
09 : (apply pickup blockA i3 i4)
10 : (apply stack blockA blockB i4 i5)
11 : (validate 1)
```

Inferring $\mathcal{M}'$ from Plan π_Λ

- Execute π_Λ in initial state I_Λ, resulting in final state $s_g = \theta(I_\Lambda, \pi_\Lambda)$.
- $\mathcal{M}'$ is given by fluents $pre_f(\xi)$, $del_f(\xi)$, and $add_f(\xi)$ that are set to true in s_g.

For each $\xi \in \mathcal{M}'$, we build the sets of preconditions, positive effects and negative effects as follows:

$$
\begin{aligned}
pre(\xi) &= \{p | pre_p(\xi) \in s_g\}_{\forall p \in \Psi_\xi}, \\
add(\xi) &= \{p | add_p(\xi) \in s_g\}_{\forall p \in \Psi_\xi}, \\
del(\xi) &= \{p | del_p(\xi) \in s_g\}_{\forall p \in \Psi_\xi}.
\end{aligned}
$$