Planning with Sensing Actions using 0-approximation

Chitta, Tuan, Xin

Department of Computer Science

Arizona State University

Action Representation

- A planning problem: P = (A,O,I,G) where
  - A: set of fluents; O: set of actions; I,G: initial and goal states
- Two types of actions: non-sensing & sensing actions
  - Non-sensing action a:
    - Precondition Pre_a: a set of fluent literals
    - Positive effects Add_a: a set of fluents
    - Negative effects Del_a: a set of fluents
  - Sensing action a:
    - Precondition Pre_a: a set of fluent literals and u-fluents u(f)
    - Sensing effects $Sens_a$: a set of fluents
- Example:

  Walk: *Pre: ~near, Add: near*   Sense-lock: *Pre: near, Sens: locked*

Types of Goals and Plan

- Two types of goals:
  - Achievement goal: making some fluents true, some fluents false;
  - Find out goal: figure out what the value for some fluents;
- Conditional Plan:
  - An empty sequence of action, [ ], is a conditional plan;
  - If a is an action then a is a conditional plan;
  - If c_1;…; c_n ($n \geq 1$) are conditional plans and □1, ..., □n are conjunction of fuent literals (which are mutually exclusive but not necessarily exhaustive), then the following is a conditional plan:
    - Case
      - □ □1 □ c1;
      - ….
      - □ □n □ cn;
  - If c1, c2 are conditional plans then c1; c2 is a conditional plan;

State Model in Progression Search

A brief reminder: (Chitta & Son AIJ'01)

- In A_k, a c-state is $<s, \square>$;
- An a-state is $\square = <T; F>$;
- Let $\square_1 = < T_1; F_1 >$ and $\square_2 = < T_2; F_2 >$ we define:
  - $\square_1 \square \square_2$ if $T_1 \square T_2$ and $F_1 \square F_2$
  - $\square_1 \setminus \square_2$ denotes $T_1 \setminus T_2 \square F_1 \setminus F_2$
  - For a set of fluents X:

    $X \setminus < T; F>$ denotes $X \setminus T \square F$
- f is true (resp. false) in $\square$ if f $\square$ T (resp. F)
- f is unknown in $\square$ if f $\square$ T $\square$ F

State Model in Progression Search

- An a-state is $\square$ = < T; F>;
- A planning problem P defines a state-space

  S =<$\square$, $\square_0$, $\square_G$, A(.), Progress, c>

  where

  - $\square$ is a set of a-state $\square$ =<T,F>;
  - The initial state $\square_0$ is the a-state I;
  - The goal states are a-states $\square\square$ $\square_G$ such that G $\square$ $\square$;
  - The actions a $\square$ A($\square$) are actions that are executable in $\square$;
  - The progression function Progress maps a pair of a-states and actions into a-states or sets of a-states;
  - All action cost c(a, $\square$) are 1.

State Model in Progression Search

- Given an a-state □ = < T; F> and any action a, we say that a is executable in □ if Pre_a holds in □.
- A transition function Progress(□; a) is defined as follows:
  - if a is not executable in □ then Progress(□; a) = □;
  - if a is executable in □ and a is a non-sensing action:

    Progress(□; a) = <T \ Del_a □ Add_a; F \ Add_a □ Del_a >

  - if a is executable in □ and a is a sensing action:

    Progress(□; a) = {□' l □ □ □' and $Sens_a$ \ □ = □' \ □}

Extended Transition Function

- The extended transition function of Progression, denoted by Progression*, which maps a pair of a-states and conditional plans into a-states, is defined as follows:
  - Progression*(□; [ ]) = □.
  - For an action a, Progression*(□; a) = Progression(□; a).
  - For p is a plan of the form:
    - Case
      - □ □1 □ c1;
      - ….
      - □ □n □ cn. . .

    Progression*(□; p) = Progression*(□; ci) if □i holds in □

    □ if none of □i holds in □
  - For p = c1;c2 where c1 c2 are conditional plans

    Progression*(□; p) = Progression*(Progression*(□,c1), c2)
  - □ Progression*(□,p) = □ for every p.

State Model in Regression Search

- P defines a regression state-space

  S =<□, $\square_0$, $\square_G$, A(.), Regress, c>

  where:

  - □ is a set of partial a-states □ = [T,F], where □ represents a set $\square_\square$={<T’, F’> | T □ T’ and F □ F’} – extension set of □;
  - The initial state $\square_0$ is the partial a-state G;
  - The goal states are partial a-state □□ $\square_G$ such that I □ $\square_\square$;
  - The actions a □ A(□) are actions that are applicable in □ ;
  - The regression function Regress maps a pair of partial a-states (or sets of partial a-states) and actions into partial a-states;
  - All action cost c(a, □) are 1.

State Model in Regression Search Cont'd

Given a partial a-state □ = [T; F].

- Let a be a non-sensing action. We say that a is applicable in □ if

  Add_a □ T≠∅ or Del_a □ F≠∅; Add_a □ F=∅; Del_a □ T=∅;

  $Pre+_a$ □ F □ Del_a ; and $Pre-_a$ □ T □ Add_a
- Let $□_1$; … ; $□_n$ be a set of distinct partial a-states which differ only on a subset of fluents $s_{a'}$ sensed by a sensing action a, where $s_{a'}$ □ $Sens_a$,

  $n = 2^{|s_{a'}|}$, and $|s_{a'}|$ is the length of sense list of action a.
- We say that a is applicable to ({$□_1$; … ; $□_n$} if

  Pre_a+ □ $□_i$.F = ∅ and Pre_a- □ $□_i$.T = ∅ , for any i

State Model in Regression Search Cont'd

Given a partial a-state □ = [T; F].

- For a non-sensing action, *Regress* maps a pair of a partial a-state and an action to another partial a-state :

  Regress(□; a) = [T \ Add_a □ Pre_a+; F \ Del_a □ Pre_a-]

- For a sensing action, *Regress* maps a set of partial a-states and an action to a partial a-state. For a sensing action *a* applicable in {$□_1$; … ; $□_n$}; and any *i* □ *{1, … , n}*:

  Regression({$□_1$; … ; $□_n$}; a) = [$□_i$.T \ $Sens_{a'}$ □ Pre_{a+}; F \ $Sens_{a'}$ □ Pre_a-]

- if a is not executable in □ then Regression(□; a) = □;

Extended Transition Function

- The extended transition function of Regression, denoted by Regression*, which maps a pair of a-states and conditional plans into a-states, is defined as follows:
  - Regression*(□; [ ]) = □.
  - For a non-sensing action a, Regression*(□; a) = Regression(□; a).
  - For a sensing action a and a set of partial a-states $\square_1$; … ; $\square_n$, Regression*({$\square_1$; … ; $\square_n$}; a) = Regression({$\square_1$; … ; $\square_n$}; a).
  - For p is a plan of the form:
    - Case
      - □1 □ c1;
      - ….
      - □n □ cn. . .

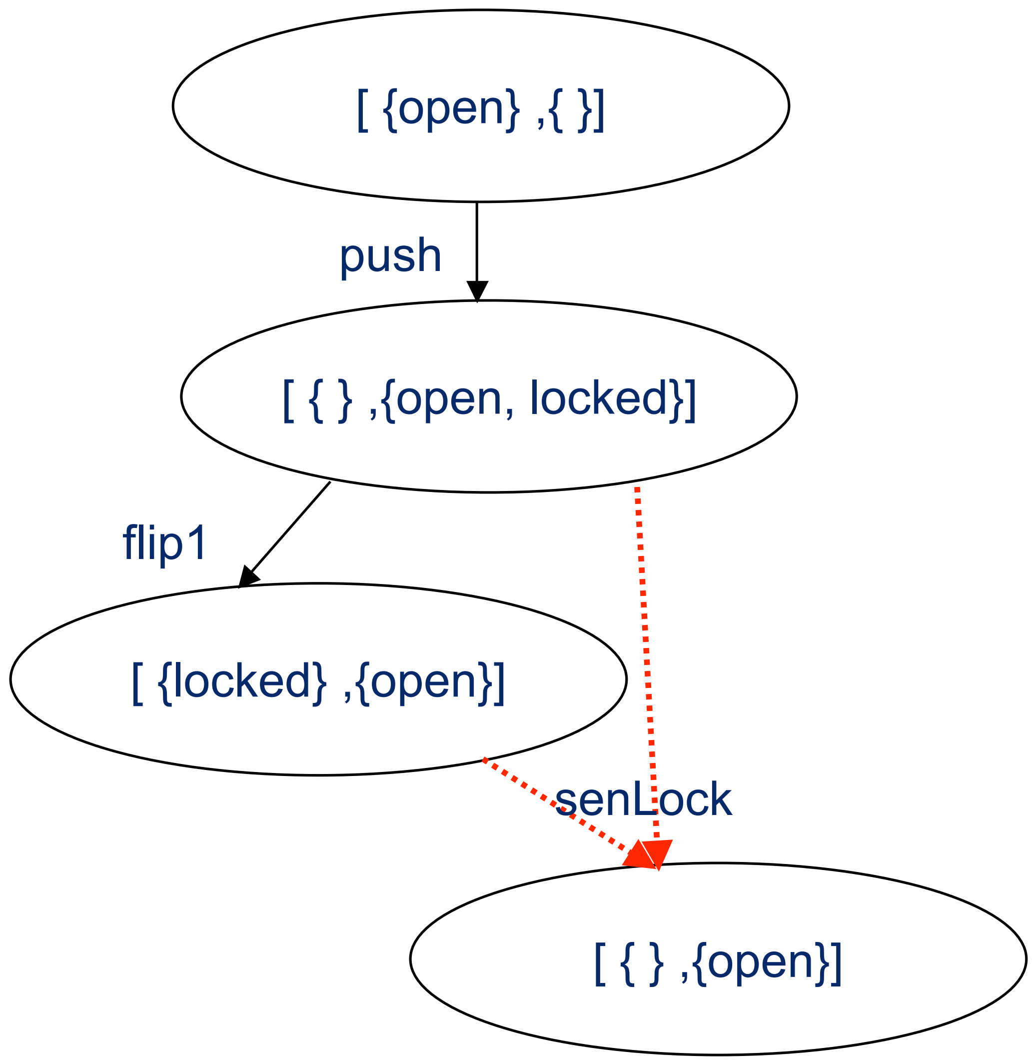
    Regression*(□; p) = {Regression*(□; c_1); … ;Regression*(□; c_n)}
    where □i is known to be true in Regression*(□; ci); (i = 1; … ; n).
  - For p = c1; c2, where c1; c2 are conditional plans,
    Regression*(□; p) = Regression*(Regression*(□; c2); c1):
  - Regression€(□; p) = □ for every plan p.

A Small Example

Fluents: open, locked

Actions:

- push:
  - Prec+: { }
  - Prec-: {locked}
  - Add: {open}
  - Del: { }
- flip1:
  - Prec+: {locked}
  - Prec-: { }
  - Add: { }
  - Del: {locked}
- flip2:
  - Prec+: { }
  - Prec-: {locked}
  - Add: {locked}
  - Del: { }
- SenLock:
  - Prec+: { }
  - Prec-: { }
  - SenList: {lock }



Lemma 1 – Regression for Non-sensing actions

- For partial a-states □ and □', and a non-sensing action a,

Regression(□; a) = □'

implies

{Progression(□''$_1$; a); … ; Progression(□''$_n$; a)} □ □$_□$

where

□$_□$ ={□$_1$,…, □$_n$} is the extension of □ , and

□$_{□'}$ = {□''$_1$,…, □''$_m$ } is the extension of □'.

Corollary

- For partial a-states □ and □', and a sequence of non-sensing actions $c = a_1;\ldots;a_n$,

Regression(□; c) = □'

implies

$\{Progression(□''_1; c); \ldots ; Progression(□''_m; c)\}$ □ $□_{□}$

where

$□_{□} = \{□_1, \ldots, □_k\}$ is the extension of □ , and

$□_{□'} = \{□''_1, \ldots, □''_m\}$ is the extension of □'.

Lemma 2 – Sensing Actions

- For a set of partial a-states {$□_1$, …, $□_n$} and a partial a-state □', and a sensing action a,

Regression({$□_1$, …, $□_n$}; a) = □'

implies for every extension □'' of □',

Progression(□'',a) □ □$□_1$ □ … □ □$□_n$

where

□$□_i$ is the extension of $□_i$.

Lemma 3 -- Plans

- For a planning problem, given initial state I and goal states G, and a conditional plan c.
- Let □ = Regression*(G; c),

I □ $□_{□}$ implies

Progression*(I; c) □ $□_{G}$

where $□_{G}$ is an extension of G.
where $□_{□}$ is an extension of □ = Regression*(G; c),

Algorithm on Regression

- S: partial a-state
- N: set of open nodes in search, each node is a partial a-state
- O: set of actions
- I: Initial state
- G: goal state
- S := I; N := G;
- Plan(I,N)
  - OldN := N;
  - if exists s □ N & I □ $\square_s$
    - then return s;
  - New := Regression(N,O);
  - N := N □ New
  - if N ≠ OldN then
    - Plan(I,N);
  - else
    - return Failure; //cannot regress further

Regression Algorithm Cont’d

- Regression(N,O)
  - M := empty;
  - for each o □ O
    - if(o is nonsensing action)
      - select s □ N where o is applicable in s;
      - s' = Regression(s,o);
      - s'.action := s.action □ o;
    - else
      - select set S = {s1 ... sn} of N where o is applicable in S;
      - s' = Regress(S,o);
      - s'.action := {s1.action + ... + sn.action} □ o;
    - if s' □ M := M □ s';
  - endForEach
  - return M;

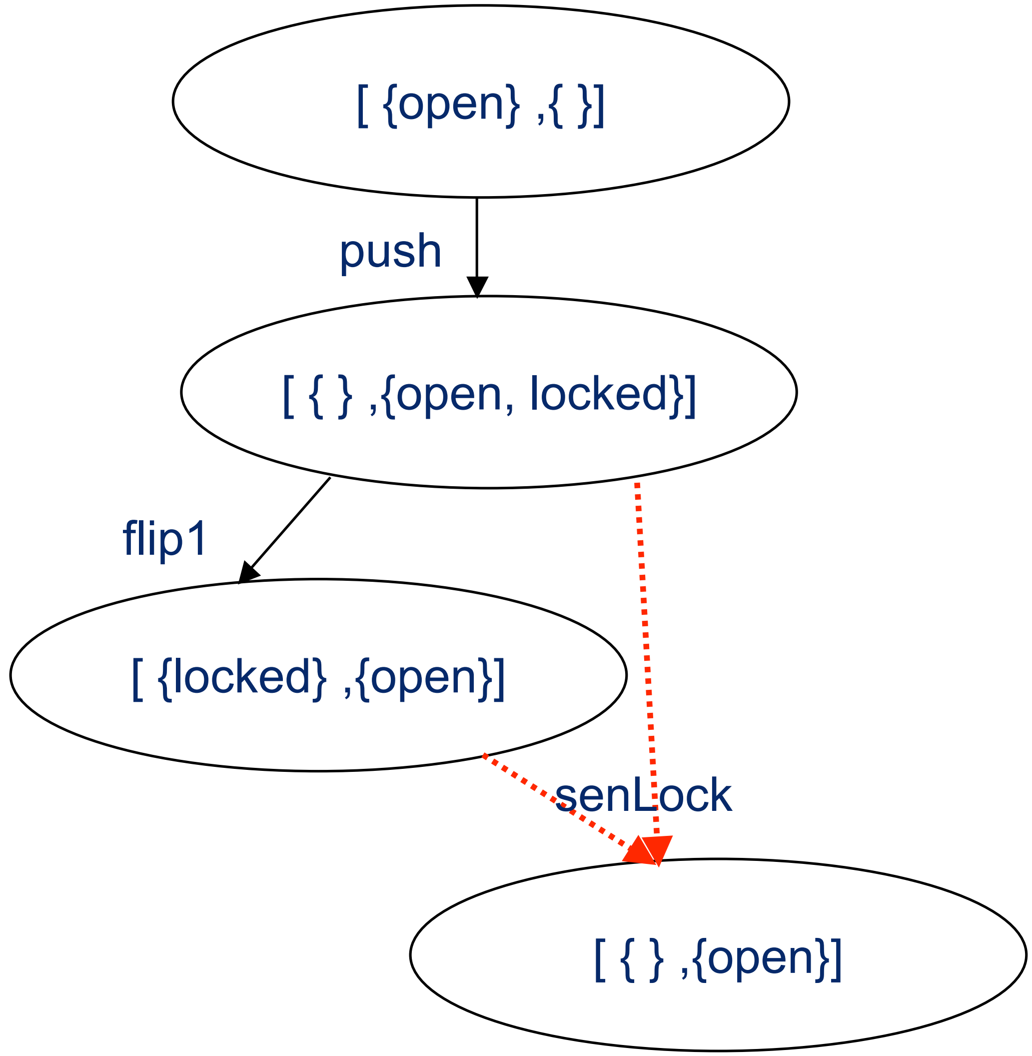
- When a plan is found, a returned state will have all actions of a plan in backward order.

A Small Example

Fluents: open, locked

Actions:

- push:
  - Prec+: { }
  - Prec-: {locked}
  - Add: {open}
  - Del: { }
- flip1:
  - Prec+: {locked}
  - Prec-: { }
  - Add: { }
  - Del: {locked}
- flip2:
  - Prec+: { }
  - Prec-: {locked}
  - Add: {locked}
  - Del: { }
- SenLock:
  - Prec+: { }
  - Prec-: { }
  - SenList: {lock }



Future Work

- Heuristics: on-going
- Implementation