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 $c_1;\ldots; c_n$ ($n \geq 1$) are conditional plans and $\square 1, \ldots, \square n$ are conjunction of fluent literals (which are mutually exclusive but not necessarily exhaustive), then the following is a conditional plan:
    - Case
      - $\square 1 \in c_1$;
      - ....
      - $\square n \in c_n$;
  - If $c_1, c_2$ are conditional plans then $c_1; c_2$ 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:
  - $\square_1 \subseteq \square_2$ if $T_1 \subseteq T_2$ and $F_1 \subseteq F_2$
  - $\square_1 \setminus \square_2$ denotes $T_1 \setminus T_2 \supseteq F_1 \setminus F_2$
  - For a set of fluents $X$:
    - $X \setminus <T; F>$ denotes $X \setminus T \supseteq F$
- $f$ is true (resp. false) in $\square$ if $f \in T$ (resp. $F$)
- $f$ is unknown in $\square$ if $f \in T \supseteq F$
State Model in Progression Search

- An a-state is \( \mathcal{A} = \langle T; F \rangle \);
- A planning problem P defines a state-space
  \[ S = \langle \mathcal{A}, s_0, S_G, A(\cdot), \text{Progress}, c \rangle \]

where
- \( \mathcal{A} \) is a set of a-state \( \mathcal{A} = \langle T, F \rangle \);
- The initial state \( s_0 \) is the a-state I;
- The goal states are a-states \( S_G \) such that \( G \in S_G \);
- The actions \( a \in A(\cdot) \) are actions that are executable in \( s \);
- The progression function Progress maps a pair of a-states and actions into a-states or sets of a-states;
- All action cost \( c(a, \mathcal{A}) \) are 1.
Given an a-state $s = <T; F>$ and any action $a$, we say that $a$ is executable in $s$ if $\text{Pre}_a$ holds in $s$.

A transition function $\text{Progress}(s; a)$ is defined as follows:

- if $a$ is not executable in $s$ then $\text{Progress}(s; a) = \emptyset$;
- if $a$ is executable in $s$ and $a$ is a non-sensing action:
  $$\text{Progress}(s; a) = <T \setminus \text{Del}_a \uparrow \text{Add}_a; F \setminus \text{Add}_a \downarrow \text{Del}_a >$$
- if $a$ is executable in $s$ and $a$ is a sensing action:
  $$\text{Progress}(s; a) = \{s' | s \in s' \text{ and } \text{Sens}_a \downarrow s = s' \downarrow \}$$
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
    - Case 1: c1;
    - ....
    - Case 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
  \[ S =<\mathcal{P}, s_0, S_g, A(.), \text{Regress}, c> \]

where:
- \( \mathcal{P} \) is a set of partial a-states \( \mathcal{P} = [T,F] \), where \( \Phi \) represents a set \( \Phi = \{<T', F'> | T \notin T' \text{ and } F \notin F'\} \) – extension set of \( \Phi \);
- The initial state \( s_0 \) is the partial a-state \( G \);
- The goal states are partial a-state \( \mathcal{P}_G \) such that \( I \notin \mathcal{P}_G \);
- The actions \( a \in A(\mathcal{P}) \) are actions that are applicable in \( \mathcal{P} \);
- The regression function \( \text{Regress} \) maps a pair of partial a-states (or sets of partial a-states) and actions into partial a-states;
- All action cost \( c(a, \mathcal{P}) \) are 1.
State Model in Regression Search Cont’d

Given a partial a-state $\mathcal{a} = [T; F]$.

- Let $a$ be a non-sensing action. We say that $a$ is applicable in $\mathcal{a}$ if
  \[ \text{Add}_a \neg T \neq \emptyset \text{ or } \text{Del}_a \neg F \neq \emptyset; \quad \text{Add}_a \neg F = \emptyset; \quad \text{Del}_a \neg T = \emptyset; \]
  \[ \text{Pre}^+_a \neg F \rightarrow \text{Del}_a; \quad \text{and } \text{Pre}^-_a \neg T \rightarrow \text{Add}_a \]

- Let $\mathcal{a}_1; \ldots; \mathcal{a}_n$ 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 $s_a' \in \text{Sens}_a$.
  \[ n = 2^{\text{Isa'}} \text{, and } \text{Isa'} \text{ is the length of sense list of action } a. \]

- We say that $a$ is applicable to $\{\mathcal{a}_1; \ldots; \mathcal{a}_n\}$ if
  \[ \text{Pre}^+_a \mathcal{a}_i \neg F = \emptyset \quad \text{and } \text{Pre}^-_a \mathcal{a}_i \neg T = \emptyset, \text{ for any } i \]
State Model in Regression Search Cont’d

Given a partial a-state \( s = [T; F] \).

- For a non-sensing action, \( \text{Regress} \) maps a pair of a partial a-state and an action to another partial a-state:
  \[
  \text{Regress}(s; a) = [T \setminus \text{Add}_a \sqcup \text{Pre}_a^{+}; F \setminus \text{Del}_a \sqcup \text{Pre}_a^{-}]
  \]

- For a sensing action, \( \text{Regress} \) maps a set of partial a-states and an action to a partial a-state. For a sensing action \( a \) applicable in \( \{s_1; \ldots; s_n\} \); and any \( i \in \{1, \ldots, n\} \):
  \[
  \text{Regression}(\{s_1; \ldots; s_n\}; a) = [s_i; T \setminus \text{Sens}_{a'} \sqcup \text{Pre}_{a'}^{+}; F \setminus \text{Sens}_{a'} \sqcup \text{Pre}_{a'}^{-}]
  \]

- if \( a \) is not executable in \( s \) then \( \text{Regression}(s; a) = s \);
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 \( s_1; \ldots; s_n \), Regression*(\{\( s_1; \ldots; s_n \}; a) = Regression(\{\( s_1; \ldots; s_n \}; a).
- For \( p \) is a plan of the form:
  - Case
    - \( 1 \) \( c_1 \);
    - ....
    - \( n \) \( c_n \). ... 
    - Regression*(∅; p) = \{Regression*(∅; c_1); \ldots; Regression*(∅; c_n)\}
      where \( i \) is known to be true in Regression*(∅; c_i); (i = 1; \ldots; n).
- For \( p = c_1; c_2 \), where \( c_1; c_2 \) are conditional plans,
  - Regression*(∅; p) = Regression*(Regression*(∅; c_2); c_1);
- 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 \( s \) and \( s' \), and a non-sensing action \( a \),

\[
\text{Regression}(s; a) = s' \\
\text{implies} \\
\{\text{Progression}(s_1'; a); \ldots; \text{Progression}(s_n'; a)\} \\ \\
\text{where} \\
\{s_1', \ldots, s_n'\} \text{ is the extension of } s' \text{, and} \\
\{s_1^{''}, \ldots, s_m^{''}\} \text{ is the extension of } s'.
\]
Corollary

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

$$\text{Regression}(s; c) = s'$$

implies

$$\{\text{Progression}(s''_1; c); \ldots; \text{Progression}(s''_m; c)\} \subseteq s'$$

where

$s'' = \{s'_1, \ldots, s'_k\}$ is the extension of $s$, and

$s''' = \{s''''_1, \ldots, s''''_m\}$ is the extension of $s'$. 
Lemma 2 – Sensing Actions

For a set of partial a-states \{s_1, ..., s_n\} and a partial a-state \(s'\), and a sensing action \(a\),

\[
\text{Regression}(\{s_1, ..., s_n\}; a) = s' \\
\text{implies for every extension } s'' \text{ of } s', \\
\text{Progression}(s'', a) \models s'_{1} \quad \cdots \quad s'_{n}
\]

where

\(s'_{i}\) is the extension of \(s_i\) .
Lemma 3 -- Plans

- For a planning problem, given initial state $I$ and goal states $G$, and a conditional plan $c$.
- Let $D = \text{Regression}^*(G; c)$,

$I \subseteq D$ implies

$\text{Progression}^*(I; c) \subseteq G_D$

where $G_D$ is an extension of $G$.

where $D$ is an extension of $D = \text{Regression}^*(G; c)$,
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 ⊆ 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.
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