Planning II

CSE 473
Logistics

- Tournament!
- PS3 - later today

Non programming exercises
Programming component: (mini project)
  - SPAM detection
    - Bag of words representation
    - Decision tree induction
    - Ensembles
    - Naïve Bayes
    - Experiments & writeup
  - Due in stages thru quarter’s end
  - New teams of 2 - due today
Generative Planning

Input
Description of (initial state of) world \((in \text{ some \ KR})\)
Description of goal \((in \text{ some \ KR})\)
Description of available actions \((in \text{ some \ KR})\)

Output
Controller
E.g. Sequence of actions
E.g. Plan with loops and conditionals
E.g. Policy = \(f: \text{ states} \rightarrow \text{ actions}\)
Simplifying Assumptions

- Static vs. Dynamic
- Instantaneous vs. Durative
- Deterministic vs. Stochastic
- Fully Observable vs. Partially Observable
- Perfect vs. Noisy
- Instantaneous vs. Durative
- Deterministic vs. Stochastic
- Full vs. Partial satisfaction

What action next?
STRIPS Action Schemata

- Instead of defining ground actions: `pickup-A` and `pickup-B` and ...
- Define a schema:

```scheme
(:operator pick-up
  :parameters ((block ?ob1))
  :precondition (and (clear ?ob1)
                      (on-table ?ob1)
                      (arm-empty))
  :effect (and (not (clear ?ob1))
            (not (on-table ?ob1))
            (not (arm-empty))
            (holding ?ob1)))
```

Note: strips doesn’t allow derived effects; you must be complete!
Outline

- Defn Planning
- STRIPS representation & assumptions
- World-space search
  - Forward chaining
  - Backwards chaining
  - Heuristics
- Graphplan
- SATplan
Backward-Chaining Search Thru Space of Partial World-States

- Problem: Many possible goal states are equally acceptable.
- From which one does one search?

Initial State is completely defined
Regression

- Let $G$ be a KR sentence (e.g. in logic)
- Regressing a goal, $G$, thru an action, $A$
- Yields the weakest precondition $G'$
  
  Such that: if $G'$ is true before $A$ is executed, $G$ is guaranteed to be true afterwards

$G'$  \[\rightarrow\] precond \[\rightarrow\]  $A$  \[\rightarrow\] effect \[\rightarrow\]  $G$

- Represents a set of world states
- Represents a set of world states

© Daniel S. Weld
Regression Example

\[
\text{pick-up} : \text{parameters}\ ((\text{block } ?\text{ob1}))
\]

\[
\text{:precondition}\ (\text{and}\ (\text{clear } ?\text{ob1})
\]

\[
(\text{on-table } ?\text{ob1})
\]

\[
(\text{arm-empty})
\]

\[
\text{:effect}\ (\text{and}\ (\text{not } \text{clear } ?\text{ob1}))
\]

\[
(\text{not } \text{on-table } ?\text{ob1}))
\]

\[
(\text{not } \text{arm-empty}))
\]

\[
(\text{holding } ?\text{ob1}))\]

\[G' \rightarrow \text{precond} \rightarrow A \rightarrow \text{effect} \rightarrow G\]

\[
(\text{and}\ (\text{clear } C))
\]

\[
(\text{on-table } C)
\]

\[
(\text{arm-empty})
\]

\[
(\text{on } A\ B))\]

Disjunction

\[\forall \text{ preconditions}\]
Conditional Effects

\[
\text{move-briefcase (?loc ?new)}
\]
\[
\text{:prec (and (at briefcase ?loc) (location ?new))}
\]
\[
(\text{not (\ = \ ?loc \ ?new\))})
\]
\[
\text{:effect (and (at briefcase ?new) (not (at briefcase ?loc)))}
\]
\[
(\text{when (in paycheck briefcase)}
\]
\[
(\text{and (at paycheck ?new)}
\]
\[
(\text{not (at paycheck ?loc)))})
\]
\[
(\text{when (in keys briefcase)}
\]
\[
(\text{and (at keys ?new)}
\]
\[
(\text{not (at keys ?loc)))})
\]
Regressing Conditional Effects

\[ G' \xrightarrow{\text{precond}} A \xrightarrow{\text{effect}} G \]

\[
(\text{and} \ (\text{at briefcase bank}) \\
(\text{in keys briefcase}) \\
(\text{not} \ (\text{in paycheck briefcase})) \\
(\text{at paycheck bank}))
\]

\[
(\text{and} \ (\text{at keys home}) \\
(\text{at paycheck bank}))
\]

move-briefcase (?loc ?new)
  :prec  (and (at briefcase ?loc)
           (not (= ?loc ?new)))
  :effect (and (at briefcase ?new) (not (at briefcase ?loc))
           (when (in paycheck briefcase)
               (and (at paycheck ?new)
                   (not (at paycheck ?loc))))
           (when (in keys briefcase)
               (and (at keys ?new)
                   (not (at keys ?loc)))))
Heuristics for Regression

• Suppose: searching backwards to init state

• Want heuristic
  Under-estimate of distance from Init state to set of states represented by $G$

????
Outline

• Defn Planning
• STRIPS representation & assumptions
• World-space search
• Graphplan
  Planning Graph Expansion
  • Mutex relations
  Solution Extraction
• SATplan
Graphplan

• Phase 1 - Graph Expansion
  Necessary (insufficient) conditions for plan existence
  Local consistency of plan-as-CSP

• Phase 2 - Solution Extraction
  Variables
    • action execution at a time point
  Constraints
    • goals, subgoals achieved
    • no side-effects between actions
The Plan Graph

Note: a few noops missing for clarity
Graph Expansion

Proposition level 0
- initial conditions

Action level i
- no-op for each proposition at level i-1
- action for each operator instance whose preconditions exist at level i-1

Proposition level i
- effects of each no-op and action at level i
Constructing the planning graph...

- **Initial proposition layer**
  Just the initial conditions

- **Action layer i**
  If all of an action’s preconds are in i-1
  Then add action to layer I
  Nop actions have P as precond and effect

- **Proposition layer i+1**
  For each action at layer i
  Add all its effects at layer i+1
Mutual Exclusion

Two actions are mutex if
• one clobbers the other’s effects or preconditions
• they have mutex preconditions

Two proposition are mutex if
• one is the negation of the other
• all ways of achieving them are mutex
Mutual Exclusion

- **Actions** $A, B$ *exclusive (at a level)* if
  - $A$ deletes $B$’s precond, or
  - $B$ deletes $A$’s precond, or
  - $A$ & $B$ have inconsistent preconds

- **Propositions** $P, Q$ *inconsistent (at a level)* if
  - all ways to achieve $P$
  - exclude all ways to achieve $Q$
Graphplan

• Create level 0 in planning graph
• Loop
  If goal $\subseteq$ contents of highest level (nonmutex)
  Then search graph for solution
    • If find a solution then return and terminate
  Else extend graph one more level

A kind of double search: forward direction checks necessary (but insufficient) conditions for a solution, ...
Backward search verifies...
Searching for a Solution

• For each goal \( G \) at time \( t \)
  
  For each action \( A \) making \( G \) true \( @t \)
  
  • Select \( A \) unless mutex with previously chosen action
  
  • If no actions work, backup to last \( G \) (breadth first search)

• Recurse on
  
  preconditions of actions selected, \( t-1 \)
Searching for a solution

If goals are present & non-mutex:
Choose action to achieve each goal
Add preconditions to next goal set

Recurse!
Dinner Date

Initial Conditions: (:and (cleanHands) (quiet))

Goal: (:and (noGarbage) (dinner) (present))

Actions:

(:operator carry :precondition
 :effect (:and (noGarbage) (:not (cleanHands))))

(:operator dolly :precondition
 :effect (:and (noGarbage) (:not (quiet))))

(:operator cook :precondition (cleanHands)
 :effect (dinner))

(:operator wrap :precondition (quiet)
 :effect (present))
Planning Graph

- noGarb
- carr
- cleanH
- dolly
- quiet
- cook
- wrap
- dinner
- present

© Daniel S. Weld
Are there any exclusions?
Do we have a solution?
Extend the Planning Graph

0 Prop 1 Action 2 Prop 3 Action 4 Prop
Searching Backwards

- noGarb
- cleanH
- quiet
- dinner
- present

- carr
- dolly
- cook
- wrap

- carry
- dolly
- cook
- wrap

0 Prop 1 Action 2 Prop 3 Action 4 Prop
One (of 4) Possibilities

0 Prop 1 Action 2 Prop 3 Action 4 Prop

© Daniel S. Weld
One (of 4) possibilities

0 Prop  1 Action  2 Prop  3 Action  4 Prop

© Daniel S. Weld
Graphplan Solution Extraction as a Constraint Network

- Not dolly & wrap
- Not carry & cook

- Present
- Dinner
- NoGarb