Overview

Backward Strips Planner

• Improvements

POP Algorithm

Example

Planning as Resolution

- Since Situation Semantics entirely expressed in logic, can use theorem prover to find situation in which goal is true
+ Representation of situation with 'do' function gives plan
- Not very efficient even with breadth-first search
+ Interaction between subgoals means a lot of backtracking

Forward Planning

- Can be used with Strips or Situation Semantics
- Need to do breadth-first search due to cycles
- Initial situation model in frontier, find worlds that can be reached by performing an action
- Can be used with Strips or Situation Semantics

Planning as Resolution

Continued

Planning Summary

Strips:
- Actions specified with preconditions, add and delete list
- Actions are not part of the logic

Situation Semantics:
- Add situation variables to derived and primitive relations
- 'do' function takes action + situation, points to new situation
- 'poss' predicate defines situations in which an action is possible
- Primitive predicates have + change axioms which define when they start and when they stop being true
- Propositional definition of situation in which an action is possible
- 'do' function takes action + situation to new situation
- 'add' situation variables to derived and primitive relations
- Situation Semantics
- Actions are not part of the logic
- Actions specified with preconditions and deplete list
- Shirts
Example Run

Example Run with goal: on(a,b) on(b,c)
stack(a,b) makes on(a,b) true
stack(a,b) has preconditions clear(a) clear(b)
clear(b) is true in W
Goal is now clear(a) on(b,c)
unstack(c,a) makes clear(a) true
has preconditions clear(c) on(c,a) both true in W
Goal is now on(b,c)
stack(b,c) makes on(b,c) true
stack(b,c) has preconditions clear(b) clear(c) both true in W

Backward Strips Planner

Can do backward search from goal
Doesn't have to blindly search for world where goal is true
Can do backward search from goal

Sussman Anomaly

Block World:

<table>
<thead>
<tr>
<th>c</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
</tbody>
</table>

- Initial world: ontable(a), on(c,a), ontable(b)
- Goal: on(a,b) and on(b,c)

- Can do backward search from goal
- Does not have to blindly search for world where goal is true
Improvement 1: Clobberers

- Let's remember why actions are added in.
  - We added an action between 9 and a with effect c.
  - We added action b to achieve c.
- More general case:
  - We might later add an action before a with effect c.
  - We might have planned for c to be true since true in initial world.
  - Any action a in partial plan has precondition c.

Clobberers:

- Don't allow subsequent actions to be clobberers.
- Let's remember why actions are added in.
  - We added in action 9 between b and a with effect c.
  - We added action b to achieve c.

Overview

- Backward Strips Planner
- Improvements
- Example
  - POP Algorithm

Interactions

- Planning for each subgoal independently is problematic.
- Subgoals can have interactions
  - Planning for a subgoal can undo planning for another goal.
- Interactions are due to precondition.
  - Previous action blocked the create() that we assumed would be true.
  - Initial move: unstack(c,b)
  - Final plan is stack(b,p), unstack(c,b), stack(a,p).
Shoe Example

• Putting on shoes and socks
  Actions: rightShoe, leftShoe, rightSock, leftSock
  Partial plan specifies rightSock before rightShoe and leftSock before leftShoe

• Note compactness by not committing to ordering

• How many linearizations are there for this plan?

Partial Order Plan:

\[
\begin{align*}
\text{Start} & : \text{Sock} \\
\text{Right} & : \text{Sock} \\
\text{Left} & : \text{Sock} \\
\text{Right} & : \text{Shoe} \\
\text{Left} & : \text{Shoe} \\
\text{Finish} & : \text{On}
\end{align*}
\]

Causal Links

• Keep track of why actions were added to plan

- Causal links imply an ordering constraint
- Causal links can add to plan and need to be maintained
- Preceded action needs initial action to be done in order
- If new action can establish causal link, add it

Example: two causal links

- \( \text{on} \) is does not violate existing causal links
- We can add a new action anywhere

- Example: new causal links

Keep track of why actions were added to plan
Partial Order Planners

- Path of least commitment
  - Don't decide ordering until necessary (make sure there is a solution)
  - Start with a simple incomplete plan
  - Find unachieved precondition
  - Add action (or reuse action already in plan) to achieve precondition
  - Add causal link
  - When clobberer (fig a), add ordering constraint to make fig b or fig c

Overview

- Backward Strips Planner
- Improvements
- Example

Totally Ordered

- Pop Algorithm
- Improvements
- Backward Strips Planner
Complete Consistent Plans

**Complete**
- Every precondition of every step achieved by some step
- A step $S_i$ achieves a precondition $c$ of step $S_j$ if $S_i \prec S_j$ and $c \in \text{effects}(S_i)$
- There is no clobbering step $S_k$ such that $\neg c \in \text{effects}(S_k)$, where $S_i \prec S_k \prec S_j$ in some linearization of the plan

- Every precondition of every step achieved by some step
- Note: must check all possible linearizations of the plan

**Incomplete**
- not all preconditions achieved

**Consistent**
- No contradictions in the ordering

**Representation**
- A plan is a data structure consisting of
  - A set of plan steps
  - Plan might be a subplan of a bigger plan
  - Some steps can perform actions in parallel
  - Why arbitrarily choose one solution over another
  - Some agents can perform actions in parallel
  - Plan might be a subplan of a bigger plan
  - Why arbitrarily choose one solution over another

- Incomplete
- Initial Plan
  - Steps
    - start:step(start,initial world state,null)
    - finish:step(finish,null,goal)
  - Orderings:
    - $\{ \text{start \ before \ finish} \}$
  - Links:
    - $\{ \}$

- Final Plan
  - Execution: Perform all steps in order
  - Causality: Events happen in a specific order
  - Satisfaction: World state changes

- Steps
  - Start
  - Have(Drill)  Have(Milk)  Have(Banana)  At(Home)
  - Sells(SM,Banana)  Sells(SM,Milk)  Sells(HWS,Drill)

- Initial Plan
  - Start
  - Have(Drill)  Have(Milk)  Have(Banana)  At(Home)
  - Sells(SM,Banana)  Sells(SM,Milk)  Sells(HWS,Drill)

- Final Plan
  - Execution: Perform all steps in order
  - Causality: Events happen in a specific order
  - Satisfaction: World state changes

- Initial Plan
  - Start
  - Have(Drill)  Have(Milk)  Have(Banana)  At(Home)
  - Sells(SM,Banana)  Sells(SM,Milk)  Sells(HWS,Drill)

- Final Plan
  - Execution: Perform all steps in order
  - Causality: Events happen in a specific order
  - Satisfaction: World state changes
Example

- Add 'buy' actions to achieve the three goals: have milk, bread & drill.
- Note ordering constraints force all actions to be after start.
- Drill has goals: have milk & bread.

\[\text{At(s), Sells(s,Drill)} \quad \text{At(s), Sells(s,Milk)} \quad \text{At(s), Sells(s,Bananas)}\]

\[\text{Buy(Drill)} \quad \text{Buy(Milk)} \quad \text{Buy(Bananas)}\]

\[\text{Have(Drill), Have(Milk), Have(Bananas), At(Home)}\]

---

Algorithm

- Always work from a consistent plan with no clobberers.
- CHOOSE an unachieved precondition.
- CHOOSE an effect of an existing step to achieve it.
- Add causal link and ordering constraint.
- OR CHOOSE a new step to achieve precondition.
- Resolve threats:
  - Remove any clobberer.
  - Add causal link & ordering constraint.
- OR CHOOSE an unmet precondition.
- Always work from a consistent plan with no clobberers.

---

Overview

- Backward Strips Planner
- Improvements
- POP Algorithm
Variables and Threats

- After adding first causal link, we have ¬at(X) as an effect and at(X) as a causal link.
- Should this be considered a threat?

Approaches:
- Add constraint \( X \neq \text{home} \), not easy in Prolog.
- Consider threat if unifiable \( \text{at}(X) = \text{at}(	ext{home}) \).
- Will this still give a complete algorithm?
- Consider threat if exact match \( \text{at}(X) = \text{at}(	ext{home}) \).
- After any operation that binds variables, need to check for clobberers.

Variables

- Only allow value to be from a specified set.
- Don't allow variables to be bound with something.
- Could improve more complicated constraints.
- Can use Prolog's variable substitutions.
- Can backtracking need to undo variable constraints? The Prolog.
- Use variable constraints. The Prolog.

When choosing an action, don't need to instantiate all variables.

Example Continued

- Add \( \text{go(hws)} \) to satisfy precondition on \( \text{buy(drill)} \) and \( \text{go(sm)} \) for \( \text{buy(milk)} \).
- Not shown here are the ¬at(X) effects of \( \text{go} \).
- Any clobberers yet?
Final Plan

Example Continued

- If we used at(home) from start node to satisfy the precondition of go(hws), we can't use it for go(sm).
- Other options for at(X) precondition of go(sm) are:
  - Introduce another go action
  - Use effect of go(hws)

- Add causal link for at(home) for go(hws).
- Add causal link for go(hws) for go(sm).
- Add causal link for at(home) for go(sm).

Example Continued
CHOOSE an unachieved precondition

CHOOSE effect of an existing step to achieve it

OR CHOOSE a new step \( S_j \) to achieve precondition

For ANY CLOBBERER \( S_c \) to any causal link \( S_i c \rightarrow S_j \)

DEMOTE: make \( S_i \) precede \( S_c \)

PROMOTE: make \( S_c \) precede \( S_j \)

For ANY COMPLETE \( S_c \) to and causal link \( S_j c \rightarrow S_i \)

CHOOSE effect of an existing step to achieve it

CHOOSE an unachieved precondition

Choice Points

Lots of choice points. If we can eliminate some

- Does order of picking unachieved preconditions matter?
- Does it matter where we resolve threats?
- Are all choice points needed for completeness?

- Less backtracking for depth-first and less breadth for breadth-first

- LOTS of choice points. If we can eliminate some

- Does it matter the order we resolve threats?
- Does order of picking unachieved preconditions matter?

Properties of Algorithm

- Algorithm has choice points

- Will depth-first planner always find plan (if there is one)?
- Will breadth-first planner always find plan?
- Could get stuck down wrong path
- Can do depth-first or breadth-first search

- Does algorithm find plan if there is one?
- Is algorithm sound and complete?
- Does it matter the order we resolve threats?
- Does order of picking unachieved preconditions matter?