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
- Can be model with Strips or Situation Semantics

Forward Planning

- No easy algorithm with the desired properties
- Requires a large number of actions in parallel
- Note: actions with the desired properties are given as part of the problem
- Many situations are needed to express in logic, can be shown

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
  - \texttt{do} function takes action + situation, points to new situation
  - \texttt{poss} predicate defines situations in which an action is possible
  - Primitive predicates have:
    - \texttt{change} axioms which state when they start being true
    - \texttt{frame} axioms which state when they continue to be true
  - Both are condition on what action just occurred
Interactions

• Planning for on(a,b) on(b,c)
  - Stack a onto b, requires unstacking c from a first
  - Stack b onto c, requires unstacking a from b first
  - Final plan is unstack(c,a) stack(a,b) unstack(a,b) stack(b,c)

• If we switch the order of the goals:
  - Stack b onto c, no preconditions
  - Stack a onto b
    + Must first have a clear, unstack c a
    + Must first have c clear, unstack b c
  - Final plan is stack(b,c) unstack(b,c) unstack(c,a) stack(a,b)

• Subgoals can have interactions
  - Subsequent actions can undo previously achieved goals
  - Planning for each subgoal individually is problematic

Example

• Block World:
  - stack(C,A) move block C from on table to ontop of A
  - unstack(C,A) move block C from ontop of A onto table

  - initial world: ontable(a), on(c,a), ontable(b)

  - Goal: on(a,b) and on(b,c)

Sussman Anomaly

• Can do backward search from goal
  - Don't have to blindly search for world where goal is true
  - Find actions to make goal true

• Strips Planner
  input: goal-list Goals, initial world W
  output: list of actions P, final world

  plan = []
  while Goals:
    G = Goals[0]
    Goals = Goals[1:]
    if G is not true in W:
      find action A whose effect achieves G
      add A to beginning of plan
      call planner on preconditions of A with W
      returning plan P' and world W'
      append P' to end of plan P, append A to end of plan P
    set W to W'
    apply add/delete list of A to W

Backward Strips Planner

- Find actions to make goal true
- Don't have to blindly search for world where goals are true
- Can do backward search from goal
Causal Links

- Keep both causal links and action list
- Record precondition action that needs it
- Causal link: if no action doesn't violate the existing causal link
- We can add a new action anywhere
- We need to open a new causal link
- We need to open a new causal link
- Example: two causal links
- Better links of my actions were added to plan

Causal Links

Improvement 1: Clobberers

- Clobberers: action in partial plan has precondition
- We might have planned for it to be true since true in initial world
- We might later add an action before it that affects c

- More general case: added action b to achieve c for a
- We add action j between b and a with effect ¬c

- Let's remember why actions are added in!
- Don't allow subsequent actions to be clobberers
- Lists remaining unexpanded actions and added in
- Work through partial plan and unexplored
- We add action in the world
- We add action in the world
- We add action in the world
- When expanded, we add action in the world

- We add both actions in the same world
- We add both actions in the same world
- We add both actions in the same world
- We add both actions in the same world

- Clobberers:
- Example
- Pop Algorithm
- Improvement
- Backward-Specific Planner

Overview
**Shoe Example**

- **Partial Order Plan:**
  - LeftSockOn
  - RightSockOn
  - LeftShoeOn, RightShoeOn

- **Totally Ordered**

- **Improvement 2: Partial Order**
  - Do we need to decide full ordering of actions when planning?
  - Constraints:
    - If new action potentially violates a causal link, add ordering constraint
    - Causal link implies an ordering constraint
    - Minimum restrictions on where actions can go
    - Why not allow new action to go anywhere? Either there are no links or
      non-reversing plans with causal links
    - Does need to check if action is true in the current world

- **Questions for this plan:**
  - How many instantiations are possible?
  - Note constraints by not committing to ordering
  - Partial order exists by not committing to ordering
**Representation**

- A plan is a data structure consisting of:
  - A set of plan steps $S_1, ..., S_n$ +
  - Each step of form \( \text{Name}: \text{step(Head, Effect, Preconditions)} \) +
  - Name bound to a unique identifier so that we can ensure we can uniquely refer to each step +
  - A set of ordering constraints, e.g. $S_i \prec S_j$, $S_i$ before $S_j$ +
  - Just keep ordering constraints of the step names +
  - A set of causal links, e.g. $S_i \rightarrow S_j$: $S_i$ achieves $c$ for $S_j$ +
  - record the purposes of each step: a purpose of $S_i$ is to achieve the precondition $c$ of $S_j$ +

**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
  - Add each action (or reuse action already in plan) to achieve precondition
  - Add causal link
  - Find unordered precondition
  - Add causal link
  - When ordered (Figure 9.8)
  - Add causal link
  - When reordered (Figure 9.8)
  - Add causal link
  - When reordered (Figure 9.8)

**Overview**

- Backward Strips Planner
- Improvements
  - ⇒ POP Algorithm
  - Example

- Partial Order Planners
- Example
  - Backward Strips Planner

**Summary**
Algorithm

- Always work from a consistent plan with no clobberers

- Choose an unachieved precondition

- Choose the effect of an existing step to achieve it
  - Add causal link and ordering constraint

- Or choose a new step $S_j$ to achieve precondition
  - Add causal link and ordering constraint

- Resolve threats
  - For any clobberer $S_c$ to any causal link $S_i \rightarrow S_j$
    - Demote: make $S_c$ precede $S_i$
    - Promote: make $S_c$ come after $S_j$

- Repeat until complete, backtrack on failure

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
    + $S_i \preceq S_j$ and $c \in \text{effects}(S_i)$
    + There is no clobberer $S_k$ such that $\neg c \in \text{effects}(S_k)$, where $S_i \preceq S_k \preceq S_j$ in some linearization of the plan
  - Note: must check all possible linearizations of the plan

- Incomplete:
  - Reason about consistency of sets of time points

Initial and Final Plans

- Initial Plan
  - $\text{steps} = (\text{start}, \text{null}, \text{initial world state}, \text{null})$
  - $\text{orderings} = \{ \text{start} \prec \text{finish} \}$
  - $\text{links} = \{}$

- Finish
  - Have(Drill)  Have(Milk)  Have(Banana)  At(Home)

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

- Final Plan: need not be fully ordered
  - Why arbitrarily choose one solution over another
  - Some agents can perform actions in parallel
  - Plan might be a subplan of a bigger plan

- Plan: Initial Plan
Example Continued

- Any changes you...
- New actions to "serve the..."...conditions of...
- Add actions to satisfy preconditions on buy(drill) and go(sm) for buy(milk)

- Not shown here are the ¬At(X) effects of go
- Any clobberers yet?

At(s), Sells(s,Drill) At(s), Sells(s,Milk) At(s), Sells(s,Bananas)

Have(Drill), Have(Milk), Have(Bananas), At(Home)

At(HWS), Sells(HWS,Drill) At(SM), Sells(SM,Milk) At(SM), Sells(SM,Bananas)

Example

Overview

- Backward Strips Planner
- Improvements
- POP Algorithm

Example
Example Continued

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

At(SM), Sells(SM,Bananas)
At(SM), Sells(SM,Milk)
At(HWS), Sells(HWS,Drill)
Have(Drill), Have(Milk), Have(Bananas), At(Home)

Finish
Go(HWS)
Buy(Drill)
Buy(Milk)
Buy(Bananas)
Go(SM)
Start
At(Home) At(Home)

Variables and Threats

• After adding first causal link, we have ¬at(X) as an effect and at(home) as a causal link.
• Should this be considered a threat?
• Approaches:
  - Add constraint X ≠ home, not easy in Prolog
  - Consider threat if unifiable at(X) = at(home)
  - Will this still give us a complete algorithm?
  - Consider threat if exact match at(X) == at(home)
• After any operation that bind variables, need to check for clobberers.

Variables

• When choosing an action, don't need to instantiate all variables.
• Only show variables that need a specific set.
• Don't choose variables to be bound with constraints.
• Can use empty variables to indicate no constraints.
• Can also use consistent sets of constraints like Prolog.
• Don't have to distinguish between move(a,b,c) and move(d,c,b).
• When choosing an action, don't need to instantiate all variables.
Properties of Algorithm

- Algorithm has choice points
  - Can do depth first or breadth first search
- Is algorithm sound and complete?
  - If there is a solution, there is a sequence of choices that will find plan
  - If there is no plan, will it stop?
    - No
- Will depth first planner always find plan?
  - Could get stuck down wrong path
    - But, since searching over partial plans, many fewer wrong paths
- Will breadth first planner always find plan (if there is one)?
  - Yes. Since partial plans, much smaller search space.

Final Plan

\[
\begin{align*}
\text{At}(\text{SM}) & \quad \text{At}(\text{Home}) & \quad \text{At}(\text{HWS}) \\
\text{Have(Milk)} & \quad \text{At}(\text{Home}) & \quad \text{Have(Ban.)} & \quad \text{Have(Drill)} \\
\text{Buy(Drill)} & \\
\text{Buy(Milk)} & \\
\text{Buy(Ban.)} & \\
\text{Go(Home)} & \quad \text{Go(HWS)} & \quad \text{Go(SM)} \\
\text{Finish} & \\
\text{At}(\text{HWS}) & \quad \text{Sells(HWS,Drill)} \quad \text{At}(\text{SM}) & \quad \text{Sells(SM,Milk)} & \quad \text{At}(\text{SM}) & \quad \text{Sells(SM,Ban.)}
\end{align*}
\]
Choice Points

CHOOSE an unachieved precondition

CHOOSE effect of an existing step to achieve it

OR CHOOSE a new step $S_j$ to achieve precondition $P_c$

- For ANY CLOBBERER $S_i$ to any causal link $S_j \rightarrow S_i$

DEMOTE: make $S_i$ precede $S_j$

PROMOTE: make $S_i$ come after $S_j$

• Lots of choice points. If we can eliminate some

- Does order of picking unachieved precondition matter?
- Does it matter the order we resolve threats?
- Are all choice points needed for completeness?
- Less backtracking for depth-first and less breadth-first for breadth-first

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