Overview

- Backward Strips Planner
  - Improvements
    - $$\text{POP Algorithm}$$
  - Example

Planning

- Needed to backtrack due to cycles
- Backward checking in action
- Changes with initial world in reverse order that can be reached by
- Needed with steps of Situation Semantics
- Can be used with steps of Situation Semantics

Planning as Resolution

- Since Situation Semantics is 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
- Start with initial world in frontier, find worlds that can be reached by
- Need to do breadth-first search, due to cycles
- Goal reached when reached goal
- Need to change function with operators
- Representation of situation with do function gives plan
- Need to show situation in which goal is true
- Since Situation Semantics is entirely expressed in logic, can use theorem prover

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
  - $$\text{do}$$ function takes action and situation, points to new situation
  - $$\text{poss}$$ predicate defines situations in which an action is possible
  - Primitive predicates have:
    - $$\text{add}$$ and $$\text{delete}$$ actions which take a new situation to a new situation
    - $$\text{do}$$ function takes action + situation to new situation
    - $$\text{add}$$ and $$\text{delete}$$ variables to derived and primitive relations
  - $$\text{frame}$$ axioms which state when they are not true
  - Propagation of situation with do function gives plan
  - Need to show situation in which goal is true

Claims are not part of the logic
- Actions specified with preconditions and delete list
- $$\text{do}$$ function takes action and situation, points to new situation
- $$\text{poss}$$ predicate defines situations in which an action is possible

Example Run

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

Sussman Anomaly

- Early on, Sussman's algorithm experiences a problem where the initial world can have conflicts with the goal.
- E.g., if the initial world is ontable(a) and the goal is on(a,b), the algorithm might incorrectly add actions that conflict with the initial world.

Backward Strips Planner

- Can do a backward search from the goal.
- Finds actions to make goal true.
- Doesn't require the initial world as a search invariant.

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), ontable(c), ontable(b)
- Goal: on(a,b) and on(b,c)
Improvement I: Clobberers

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

Clobberers:

- Don't allow subsequent actions to be clobberers.
  - We can't add a new action which makes a precondition false.

Russell and Norvig, Artificial Intelligence: A Modern Approach, 1995 has an excellent writeup of this approach.

Overview

- Backward Strips Planner
- Improvements
- POP Algorithm
- Example

Interactions

- Planning for each subgoal independently is problematic.
  - Subgoals can have interactions.
  - Subsequent actions can undo previous achieved goals.
- Interactions can be problematic.
  - Previous action blocked the current one we assumed would be true.
  - Example: Move: unstack(c), and we assumed it would be true.
  - Final plan is: stack(p), stack(c)
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:

- Start
- Sock
- Right
- Shoe
- Right
- Sock
- Left
- Shoe
- Left
- Finish

Improvement 2: Partial Order

• Do we need to decide full ordering of actions when planning?
• For forward planners (initial word to goal state)
  - We needed to check if action is true in the current world
• For regressive planners with causal links
  - Why not allow new actions to go anywhere, not always at front
  - We needed to check if action is true in the current world
  - We need to respect plans with causal links
• Do we need to decide full ordering of actions when planning?

Causal Links

- Keep both causal links and action list
  - Causal links determine which actions need to be done
  - We can add a new action anywhere
    - We can add a new action anywhere
    - We can add a new action anywhere
  - Example: two 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
  - When clobberer (fig a), add ordering constraint to make fig b or fig c

![Diagram of Partial Order Planners](image)

**Overview**

- Example
- = POP Algorithm
- Improvements
- Backward Strips Planets

**Totally Ordered**

![Diagram of Totally Ordered](image)
Complete Consistent Plans

- No contradictions in the ordering
- Consistent
  • not all preconditions achieved
- Incomplete
  • Reason for consistency of set of time points
  • More traceable possible information of the plan
  • Plan is a disjunction of steps, such that $x \in \text{ effects}(s)$ where $s \preceq t \preceq t'$
  • A step $t'$ achieves a precondition of step $t$ if every precondition of every step achieved by some step
  • Every precondition of every step achieved by some step

Complete

Initial and Final Plans

Initial Plan
- Steps
  
  \[
  \begin{align*}
  \text{start} &: \text{step}(\text{start}, \text{initial world state}, \text{null}) \\
  \text{finish} &: \text{step}(\text{finish}, \text{null}, \text{goal})
  \end{align*}
  \]

- Orderings:
  \{ start \prec finish \}

- Links:
  \{
  \}

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

A plan is a data structure consisting of:

- A set of plan steps $S_i \ldots S_n$
  - Each step of form $\text{Name}: \text{step}(\text{Head}, \text{Effect}, \text{Preconditions})$
  - Name bound to a unique identifier so that we can uniquely refer to each step
- A set of ordering constraints $S_i \prec S_j$
  - Just keep ordering constraints of the step names
- A set of causal links $S_i \rightarrow S_j$
  - $S_i$ achieves $c$ for $S_j$
  - Record the purposes a step may achieve

Representation

Initial Plan

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

Final Plan

Steps

\{
  \}

Links: (plan \rightarrow finish)

Orderings (initial \rightarrow finish)

Initial and Final Plan
Example

Overview

Algorithm

- Always work from a consistent plan with no clobberers
- Resolve threats
  - Add causal link and ordering constraint
  - Add clobberer to plans to achieve precondition
- OR CHOOSE a new step 5′ to achieve precondition
- CHOOSE a new step 5′ to achieve precondition
- CHOOSE an unexpected precondition
- Always work from a consistent plan with no clobberers

- Replan until complete. Backtrack on failure
- REMOVE plans 5′ to come closer
- REMOVE plans 5′ to come closer
- POP Adjustment
- Improvements
- Backward Strips Planner
- Backward Strips Planner
- Backward Strips Planner
- Backward Strips Planner
- Backward Strips Planner

- Add causal links from initial action to be after step
- Note ordering constraints force all drill items goals: have milk, bread, drink
- Add any actions to achieve the

Example
After adding first causal link, we have \[ \neg \text{at}(X) \] as an effect and \[ \text{at}(X) \] 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 \[ \text{at}(X) = \text{at}(\text{home}) \]
- Will this still give us a complete algorithm?
- Consider threat if exact match \[ \text{at}(X) == \text{at}(\text{home}) \]

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 imagine more complicated constraints
    - Can use Prolog's variable substitutions
    - On backtracking need to undo variable constraints the Prolog
    - Use variable constraints like Prolog
      - Don't have to distinguish between unordered and unordered, etc.
      - Can resolve (\text{move}(A,B,C)) to \text{move}(A,B,C)
        - But don't care whether is \text{move}(A,B,D)
  - Example:
    - When choosing an action, don't need to instantiate all variables

Variables

Any clobberers yet?

Not shown here are the \[ \neg \text{at}(X) \] effects of \[ \text{go} \]
Example Continued

- If we used \( \text{at(Home)} \) from start node to satisfy the precondition of \( \text{go(HWS)} \), we can't use it for \( \text{go(SM)} \).

- Other options for \( \text{at(X)} \) precondition of \( \text{go(SM)} \) are:
  - Introduce another \( \text{go} \) action
  - Use effect of \( \text{go(HWS)} \)

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

\[ \text{At(Home)} \]
\[ \text{At(HWS)} \]
\[ \text{Start} \]
\[ \text{Finish} \]

- Add causal link for \( \text{at(Home)} \) for \( \text{go(HWS)} \). Any clobberers?
- Add causal link for \( \text{at(Home)} \) for \( \text{go(SM)} \). Any clobberers?

Example Continued
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

For any clobberer $S_c \rightarrow S_i$

- DEMOTE: make $S_c$ precede $S_i$
- PROMOTE: make $S_c$ follow $S_i$

For ANY new step $S_j$

- OR CHOOSE a new way step to achieve precondition
- CHOOSE an unachieved precondition

Lots of choice points. If we can eliminate some
- less backtracking for depth-first and less breadth for breadth-first

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

Properties of Algorithm

- Algorithm has choice points
- Yes. Since partial plans much smaller search space.

DEMO: make $S_c$ precede $S_i$

PROMOTE: make $S_c$ come after $S_i$

Will depth-first planner always find plan (if there is one)?
- Yes, since partial plans much smaller search space.
- Will breadth-first planner always find plan (if there is one)?
- No.
- If there is no plan, will it stop?
- No.
- Is algorithm sound and complete?
- Yes, since partial plans much smaller search space.
- Can depth-first or breadth-first search

Algorithm has choice points