Overview

• Backward Strips Planner

• POP Algorithm

• Improvements

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

- Start with initial world in frontier, find worlds that can be reached by performing an action

- Can be used with breadth-first search

Planning as Resolution

Continued

Planning Summary

• Strips:

- Actions specified with preconditions, add and delete list

- Actions are not part of the logic

- Actions specified with preconditions and delete list

• 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 being true

- change axiom specifies when they start being true

- frame axioms which define when they continue to be true

- Frame axioms specify conditions on what action just occurred

- do function takes action + situation + action, points to new situation

- Add situation variables to derived and primitive relations

- Situation Semantics

- Actions are not part of the logic

- Action specified with preconditions and delete list

- Strips:
Interactions

• Planning for each subgoal individually is problematic.
  • Subgoals can have interactions.
  • Transitive Plan is Necessity (Necessity(q) Necessity(e)(q))(p(q)).
    • Thus, if there is a clear, unstack c from a.
    • Must first have a clear, unstack e from a.
    • stack a onto b.
    • stack b onto 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 from a.
    • Must first have c clear, unstack b from 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.
    • Subsequent actions can undo previously achieved goals.
    • Planning for each subgoal individually is problematic.

Example

Block World:
  • Stack (A,C): move block C from on top of A onto table
  • Unstack (A,C): move block C from top of A onto table

Initial World:
  - ontable(a), on(c,a), ontable(b)

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

Backward Strips Planner

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

Plan = [
    if G is not true in W
      find action A whose effect achieves G
      add A to end of plan P
    call Planner on preconditions of A with W
      returning plan P’ and world W’
    append P’ to end of plan P
    set W to W’
    apply add/delete list of A to W
  ]
Causal Links

- Keep track of why actions were added to plan

Causal Link

\[ \text{on}(a,b) \]
\[ \text{clear}(a) \]
\[ \text{clear}(b) \]
\[ \text{ontable}(a) \]

- Example: two causal links

Improvement 1: Clobberers

- Clobberers:
  - Say action \( a \) in partial plan has precondition \( c \)
  - We might have planned for \( c \) to be true since true in initial world
  - We might later add an action before \( a \) with effect \( \neg c \)

- More general case:
  - We added action \( b \) to achieve \( c \) for \( a \)
  - We add in action \( j \) between \( b \) and \( a \) with effect \( \neg c \)

- Don't allow subsequent actions to be clobberers
  - Let's remember why actions are added in!

POP Algorithm

Example

- Pop a fragment
- Improvements = Backward-Skip Planner

Overview
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?

Totally Ordered

- Improvement 2: Partial Order
- Do we need to decide full ordering of actions when planning?
- Partial order plans are more compact by not committing to ordering
- Minimum restrictions on where actions can go
- Partial order plans with causal links
- If a new action potentially violates a causal link, add ordering constraint
- Global hints impose an ordering constraint
- Partial order plans with causal links
- If new action potentially violates a causal link, add ordering constraint
- Expand restrictions on where actions can go
- Do we need to decide full ordering of actions when planning?
**A plan is a data structure consisting of**

- A set of plan steps: \( S_1, ..., S_n \)
  - Each step of form \( \text{Name:step(Head,Effect,Preconditions)} \)
  - 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 \( S_j \)
  - Keep ordering constraints of the step names
- A set of causal links, e.g. \( S_i \rightarrow c \rightarrow S_j \): \( S_i \) achieves \( c \) for \( S_j \)
  - Record 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
  - Find unordered precondition
  - Don’t decide ordering until necessary
  - Add action (or reuse action already in plan) to achieve precondition
  - When deadlock (hit a dead end),
    - Add causal link
    - Add action (or reuse action already in plan) to achieve precondition

---

**Overview**

- Example
  - PO Algorithm = POP Algorithm
- Improvements
- Backward Strip Planner

---
Algorithm

- Start with a complete backward plan.
- For any clobberer $S$, add causal link $S \rightarrow S'$ and ordering constraint.
- Choose an unachieved precondition $P$.
- Choose effect of an existing step to achieve $P$.
- Add causal link and ordering constraint.
- Choose a new step $S_j$ to achieve $P$.
- Add causal link and ordering constraint.
- Resolve threats:
  - Any clobberer $S_c$ to any causal link $S_i \rightarrow S_j$.
  - Make $S_c$ precede $S_i$.
  - 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 $S_i \prec S_j$ and $c \in \text{effects}(S_i)$.
  - There is no clobberer step $S_k$ such that $c \in \text{effects}(S_k)$, where $S_i \prec S_k \prec S_j$ in some linearization of the plan.
- Incomplete:
  - Not all preconditions achieved.
  - Reason about consistency of set of time points.

Initial and Final Plans

- Initial Plan:
  - Steps $\text{start:step(start, initial world state, null)}$
  - $\text{finish:step(finish, null, goal)}$
  - Orderings: $\{\text{start} \prec \text{finish}\}$
  - Links: $\{}$
  - Start:
    - Have(Drill)
    - Have(Milk)
    - Have(Banana)
    - At(Home)
  - Finish:
    - Sells(SM, Banana)
    - Sells(SM, Milk)
    - Sells(HWS, Drill)
- Final plan: need not be fully ordered.
- Why choose one solution over another.
- Some agents can perform actions in parallel.
- Plan might be a subplan of a bigger plan.
Example Continued

- Add effects of go
- No shown here are the ¬at(X) effects of go
- Add go(hws) to satisfy precondition on buy(drill)
- and go(sm) for buy(milk)

CSE560 Class 15: 20

Example

- Add causal links from initial state to sells preconditions
- Note ordering constraints force all actions to be after start
- These goals: have milk, bread & drill
- Add buy actions to achieve the

CSE560 Class 15: 19

Overview

• Backward Strips Planner
• POP Algorithm
• Improvements

CSE560 Class 15: 18
Example Continued

Variables and Threats

- Only allow values to be from a specified set.
- Don't allow variables to be bound to something
- Could imagine more complicated constraints
- Can use Prolog's variable substitutions
- Can backtracking need to undo earlier constraints. The Prolog
  Use enable constraints. See Physical
  Don't have to distinguish between move(a,b,c) and move(a,c,b).
  When choosing an action, don't need to instantiate all variables.

Variables

Variables

- Only allow values to be from a specified set.
- Don't allow variables to be bound to something
- Could imagine more complicated constraints
- Can use Prolog's variable substitutions
- Can backtracking need to undo earlier constraints. The Prolog
  Use enable constraints. See Physical
  Don't have to distinguish between move(a,b,c) and move(a,c,b).
  When choosing an action, don't need to instantiate all variables.

Variables

Variables

- Only allow values to be from a specified set.
- Don't allow variables to be bound to something
- Could imagine more complicated constraints
- Can use Prolog's variable substitutions
- Can backtracking need to undo earlier constraints. The Prolog
  Use enable constraints. See Physical
  Don't have to distinguish between move(a,b,c) and move(a,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.

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)
- Other options for act(trasportation of 80cm) are
  - If we used (achieve) from start node to satisfy the precondition
  - Lose effect of 80cm
CHOICE POINTS

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

PRUNE: make $S_c$ come after $S_i$
DENOTE: make $S_c$ precede $S_i$
FOR ANY CLOBBERER $S$ to any causal link $S_i \rightarrow S_c$,
OR CHOOSE a new unachieved precondition
CHOOSE an unachieved precondition

• Lots of choice points. If we can eliminate some

Note: make $S_c$ come after $S_i$