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
  - \textit{do} function takes action + situation, points to new situation
  - \textit{poss} predicate defines situations in which an action is possible
  - Primitive predicates have
    + \textit{change} axioms which state when they start being true
    + \textit{frame} axioms which state when they continue to be true
    + Both are conditioned on what action just occurred
• 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
  - Need to do breadth-first search, due to cycles
Overview

⇒ Backward Strips Planner
• Improvements
• POP Algorithm
• Example
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

• Strips Planner

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

set Goals to goal, but exclude anything that is in W
set Plan to empty list
while Goal not empty
  remove first goal g from Goals
  pick an action a whose add list includes g
  add a to beginning of plan
  add preconditions of a not in initial world to beginning of G
Example

• Block World:
  \begin{itemize}
  \item stack(C,A) move block C from on table to ontop of A
  \item unstack(C,A) move block C from ontop of A onto table
  \end{itemize}
  - initial world: ontable(a), on(c,a), ontable(b)

• Goal: on(a,b) and on(b,c)

Sussman Anomoly
Example Run

while Goal not empty
    remove first goal \( g \) from Goals
    pick an action \( a \) whose add list includes \( g \)
    add \( a \) to beginning of plan
    add preconditions of \( a \) not in initial world to beginning of G

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
        has preconditions clear(b) clear(c) both true in \( W \)
Interactions

- Final plan is stack(b,c) unstack(c,a) stack(a,b)
  - illegal move: unstack(c,a)
  - Previous action clobbered the clear(c) that we assumed would be true

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

• Backward Strips Planner
⇒ Improvements
• POP Algorithm
• Example
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 $\neg c$

• Lets remember why actions are added in!
  - Don’t allow subsequent actions to be clobberers

Causal Links

• Keep track of why actions were added to plan

• Example: two causal links
  - *stack* added to plan to achieve \( \text{on}(a,b) \)
  - *clear*(a) of \( \text{on}(a,b) \) to be done by initial

• We can add a new action *anywhere*, as long as doesn’t violate existing causal links

• Causal links:
  - Give actions a name (e.g. \( a_1, a_2 \), etc)
  - Record precondition, action that needs it & action that establishes it

• Keep both causal links and action list
Improvement 2: Partial Order

• Do we need to decide full ordering of actions when planning?

• For forward planners (initial world 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, rather than always at front

• Let’s include ordering constraints
  - Minimum restrictions on where actions can go
  - Causal link implies an ordering constraint
  - If new action potentially violates a causal link, add ordering constraint
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

Total Order Plans:

Start → Right Sock → Left Sock → Right Shoe → Left Shoe → Finish

Start → Right Sock → Left Sock → Right Shoe → Left Shoe → Finish

Start → Left Sock → Right Sock → Left Shoe → Right Shoe → Finish

Start → Right Sock → Left Sock → Right Shoe → Left Shoe → Finish

Start → Left Sock → Right Sock → Left Shoe → Right Shoe → Finish
Overview

• Backward Strips Planner
• Improvements
  ⇒ POP Algorithm
• Example
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

© P. Heeman, 2020  15 of 29  CS560 Class 16: POP Algorithm
A plan is a *data structure* consisting of
- A set of plan steps $S_1...S_n$
  + Each step of form `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 $S_j$
  + Just keep ordering constraints of the step names
- A set of causal links, e.g. $S_i \rightarrow c S_j$: $S_i$ achieves $c$ for $S_j$
  + record the purposes of each step: a purpose of $S_i$ is to achieve the precondition $c$ of $S_j$
Initial and Final Plans

• Initial Plan
  - Steps
    start:step(start,initial world state,null)
    finish:step(finish,null,goal)
  - 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
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
  - Note: must check all possible linearizations of the plan
  - Reason about consistency of set of time points

• **Incomplete:**
  - not all preconditions achieved

• **Consistent:**
  - No contradictions in the ordering
Algorithm

- Always work from a consistent plan with no clobberers
- **CHOOSE** an unachieved precondition
- **CHOOSE** 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
Overview

• Backward Strips Planner
• Improvements
• POP Algorithm

⇒ Example
Example

• Add ‘buy’ actions to achieve the three goals: have milk, bread & drill
  - Note ordering constraints force all actions to be after start

• Add causal links from initial state to ‘sells’ preconditions
- Add \( \text{go}(\text{hws}) \) to satisfy precondition on \( \text{buy}(\text{drill}) \) and \( \text{go}(\text{sm}) \) for \( \text{buy}(\text{milk}) \)
- Not shown here are the \( \neg \text{at}(X) \) effects of \( \text{go} \)
- Any clobberers yet?
Variables

• When choosing an action, don’t need to instantiate all variables
  - Example:
    Choose move(A,B,C) to achieve on(a,X,c), but don’t care what B is
    Wait with instantiating B until we have to
    Hence, fewer action instances to consider
    Don’t have to distinguish between move(a,d,c) and move(a,e,c), etc

• Use variable constraints, like Prolog
  - On backtracking need to undo variable constraints, like Prolog
  - Can use Prolog’s variable substitutions

• Could imagine more complicated constraints
  - Don’t allow variable to be bound with something
  - Only allow value to be from a specified set
Variables and Threats

- After adding first causal link, we have $\neg at(X)$ as an effect and $at(X)$ as a causal link

- Should this be considered a threat?

- Approaches:
  - Add constraint $X \neq home$, 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
- Add causal link for \textit{at(home)} for \textit{go(hws)}. Any clobberers?
- Add causal link for \textit{at(home)} for \textit{go(sm)}. Any Clobberers?
• If we used $at(\text{home})$ from start node to satisfy the precondition of $go(\text{hws})$, we can’t use it for $go(\text{sm})$

• Other options for $at(X)$ precondition of $go(\text{sm})$ are
  - Introduce another $go$ action
  - Use effect of $go(\text{hws})$
Final Plan

Start

At(Home)

Go(HWS)

At(HWS) Sells(HWS, Drill)

Buy(Drill)

At(HWS)

Go(SM)

At(SM) Sells(SM, Milk) At(SM) Sells(SM, Ban.)

Buy(Milk) Buy(Ban.)

At(SM) Go(Home)

Have(Milk) At(Home) Have(Ban.) Have(Drill)

Finish

© Artificial Intelligence: A Modern Approach, Russell & Norvig

© P. Heeman, 2020
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.
Choice Points

**CHOOS**E an unachieved precondition

**CHOOS**E effect of an existing step to achieve it

**OR CHOOSE** a add new step $S_j$ to achieve precondition

For **ANY CLO**BBER**ER** $S_c$ to any causal link $S_i \rightarrow S_j$

- **DEMOT**E: make $S_c$ precede $S_i$
- **PROMOT**E: make $S_c$ come after $S_j$

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

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