

---

## 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

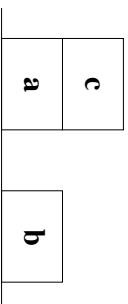
while{Goals != {}}
  Take first goal G off of Goals
  if G is not true in current world
    find action A whose effect achieves it
    call planner on preconditions of A with W
      returning plan P' and world W'
    append P' to end of plan P
  append A to end of plan P
  set current world W to W'
```

---

## 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

---

## 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

---

## Overview

---

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

---

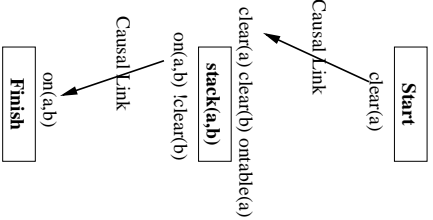
## Improvement 1 : Clobberers

---

- 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  $on(a,b)$
  - *clear(a)* of  $on(a,b)$  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. **a1**, **a2**, etc)
  - Record precondition, action that needs it & action that establishes it
- Keep both causal links and action list

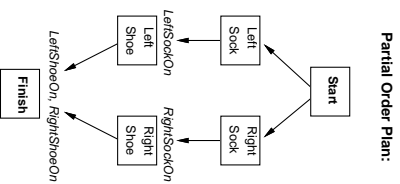


## Improvement 2: Partial Order

- Do we need to decide full ordering of actions when planning?
- For forward planners (initial 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, 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

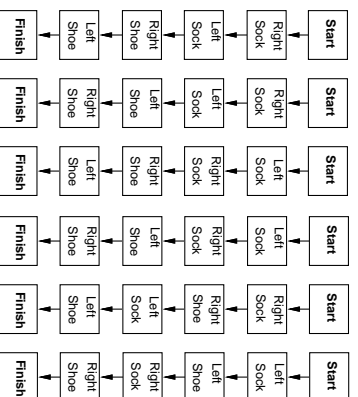
- Putting on shoes and socks
  - Actions: rightShoe, leftShoe, rightSock, leftSock
  - Partial plan specifies rightSock before rightShoe and leftSock before leftShoe
- Note compactness by not committing to ordering
- How many linearizations are there for this plan?



©Artificial Intelligence: A Modern Approach, Russell & Norvig

## Totally Ordered

Total Order Plans:



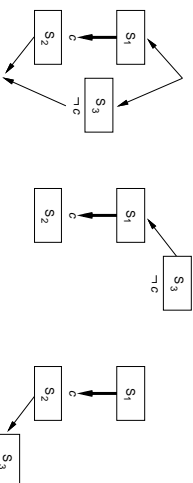
©Artificial Intelligence: A Modern Approach, Russell & Norvig

## 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



## Representation

- A plan is a *data structure* consisting of
  - A set of plan steps  $S_1 \dots S_n$ 
    - + 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.  $S_i \xrightarrow{c} S_j$ :  $S_i$  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

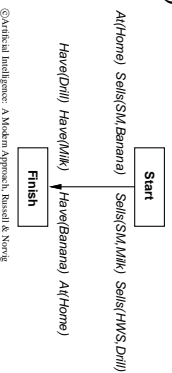
- Steps

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  $c \in effects(S_i)$
    - + There is no *clobbering* step  $S_k$  such that  $\neg c \in effects(S_k)$ , 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

---

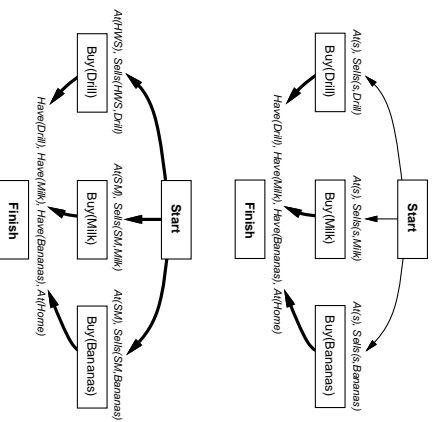
- Always work from a consistent plan with no clobberers
- **CHOOSE** an unachieved precondition
- **CHOOSE** effect of an existing step to achieve it
  - Add causal link and ordering constraint
- **OR CHOOSE** a new step  $S_j$  to achieve precondition
  - Add causal link and ordering constraint
- Resolve threats
  - For **ANY CLOBBERER**  $S_c$  to any causal link  $S_i \xrightarrow{c} S_j$ 
    - + **DEMOTE**: make  $S_c$  precede  $S_j$
    - + **PROMOTE**: make  $S_c$  come after  $S_j$
- Repeat until complete, backtrack on failure

## Overview

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

## Example

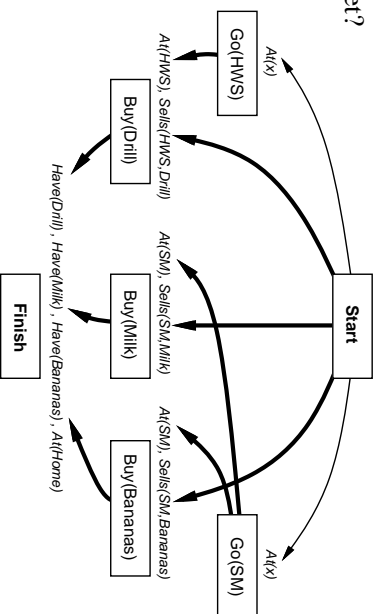
- Add 'buy' actions to achieve the three goals: have milk, bread & drill
- Note ordering constraints force all actions to be after start



- Add causal links from initial state to 'sell's' preconditions

## Example Continued

- Add *go(hws)* to satisfy precondition on *buy(drill)* and *go(sm)* for *buy(milk)*
- Not shown here are the  $\neg at(X)$  effects of *go*
- Any clobberers yet?



©Artificial Intelligence: A Modern Approach, Russell & Norvig

## Variables

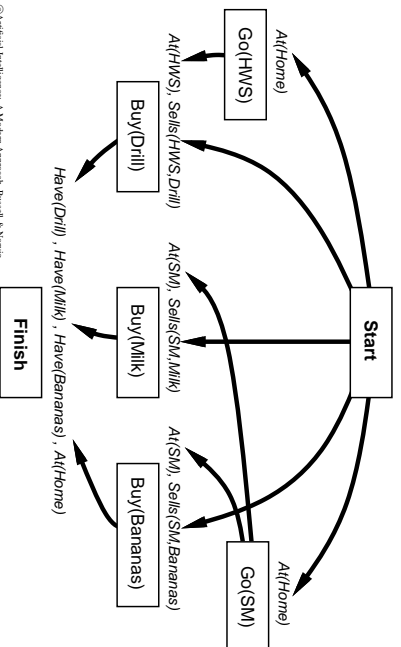
- When choosing an action, don't need to instantiate all variables
  - Example:
    - Choose *move(A,B,C)* to achieve on(*a,X,c*), but don't care what *B* is
    - Wait with instantiating *B* until we have to
    - Hence, fewer action instances to consider
    - Don't have to distinguish between *move(a,d,c)*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

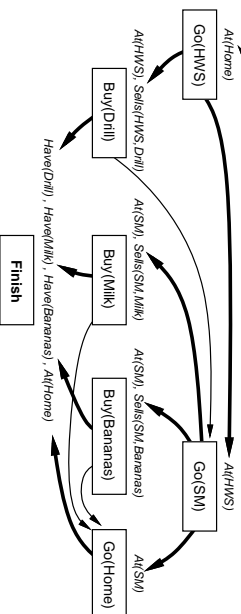
## Example Continued

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



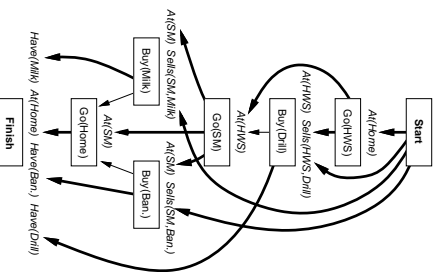
## Example Continued

- 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)*



© Artificial Intelligence: A Modern Approach, Russell & Norvig

## Final Plan



© Artificial Intelligence: A Modern Approach, Russell & Norvig

## 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

---

**CHOOSE** an unachieved precondition

**CHOOSE** effect of an existing step to achieve it

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

For **ANY Clobberer**  $S_c$  to any causal link  $S_i \xrightarrow{c} S_j$

**DEMOTTE**: make  $S_c$  precede  $S_i$

**PROMOTE**: 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 choice points needed for completeness?
  - Does it matter the order we resolve threats?
  - Does order of picking unachieved preconditions matter?