Title: Basic planning

AIMA: Chapter 11 (Sections 11.1–11.6)

Advanced Artificial Intelligence

CSCE 976, Spring 2002

www.cse.unl.edu/~choueiry/S02-976

Berthe Y. Choueiry

choueiry@cse.unl.edu, (402)472-5444

Glossary:

Given: Monday, Jan 14, 2002.

Due: Wed Jan 23, 2000

- Add list
- Causal link
- Consisten (plan)
- Complete (plan)
- Delete list
- Promotion, demotion
- Modification operator
- Negation as failure
- Progression and regression
- Refinement operator
- Situation Calculus
- STRIPS
- Theorem Proving
- Threat, clobber
- UCPOP

Outline

- Problem solving
- Situation Calculus
- STRIPS: action, operator schema
- Progression, regression planner
- Partial Order Planner (POP)

Problem

Given: Initial conditions, goals, possible actions

Find: sequences of actions that achieves specified goals

Assumptions: *Courtesy Marie desJardins*

- Atomic time: each action is indivisible
- no concurrent actions allowed, but actions need not be totally ordered
- Deterministic actions: output completely determined, no uncertainty in effects
- Agent is sole cause of change in the world
- Agent is omniscient: has complete knowledge of the state of the world
- Closed world assumption: everything known to be true in the

world is included in the state description. Anything not listed is false.

Outline:

A Planning agent, like a problem-solving agent:
— builds plans to achieve goals, then
— executes them

Planning differ from problem-solving:
— uses logical representations for $\begin{cases}\text{goals}\\ \text{states}\\ \text{actions}\end{cases}$
— the way solutions are represented: unbroken sequences
— the way search is conducted

Partial-order planning (POP) algorithm:
— finds one solution
— is complete and sound

POP is flexible enough to handle complicated domains

Problem solving: representation/design decisions

Actions : represented by programs that generate successor-state descriptions

States:

(1) all state descriptions are complete

— a complete description of the initial state

— Programs of actions generate complete descriptions of new states

(2) typically, a state is a simple data structure (*e.g.*, 8-puzzle, n-queens, etc.)

(3) used to generate a successor, evaluate a heuristic function, and test goal

Goals: big picture only (*i.e.* test goal + heuristic function) black-box, no detail

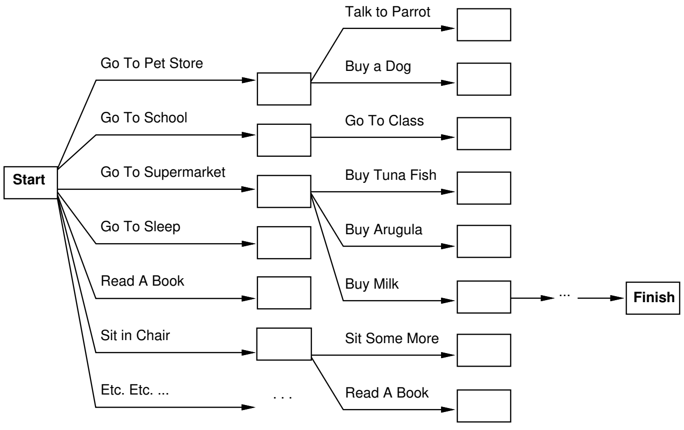
Plans: a solution is a sequence of actions, 'linearly' constructed

Shortcomings?

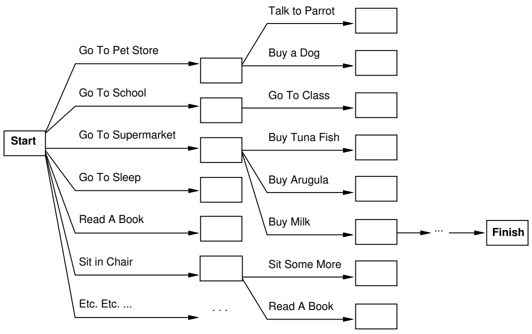
Example: Shopping

"Get a quart of milk, a bunch of bananas and a cordless drill."

Initial state: "at home without any of these objects"
Operator set: all things the agent can do.
Heuristic function: # things not yet acquired



Branching factor? depth of solution?



Heuristic function: $\left\{ \begin{array}{l} \text{chooses the states closer to the goal} \\ \text{cannot \underline{eliminate} actions from consideration} \end{array} \right.$

Starting from initial state:

$\left. \begin{array}{l} \text{(1) Go places then} \\ \text{(2) acquire objects} \end{array} \right\}$ vs. $\left\{ \begin{array}{l} \text{(1) Want to acquire objects, so} \\ \text{(2) target places where things can be acquired} \\ \quad \text{(don't go to school or sleep)} \end{array} \right.$

Planning vs. problem solving (I)

1. "Open up the representation of states, goals and actions"
2. Add actions as they are needed, relax requirement for sequential construction of solutions
3. Decomposability: develop subplans independently, then merge them

Planning vs. problem solving (II)

1– "Open up the representation of states, goals and actions"

States and goals: sets of FOL sentences

Actions: logical descriptions of preconditions and effects

Example: Goal includes $Have(Milk)$, then an action $Buy(x)$ is relevant but not $ReadaBook$

2– Add actions as they are needed, do not concentrate on completing a partial solution

Consider $Buy(Milk)$ even before choosing from where to buy it, how to get there, etc.

State description need not be complete: $At(Supermarket)$ with or without bananas, drill, etc..

Planning vs. problem solving (III)

3– Decomposability: most parts of the world are independent of most other parts

Example

Subplan 1: "get milk and bananas" (*e.g.*, go to Hy-Vee)

Subplan 2 "get a drill" (*e.g.*, ask your neighbor)

then put subplans together

— Works as long as subplans do not interfere (*e.g.*, out of money/time, shops closed)

— How about the 8-puzzle problem?

cost of combining solutions too high

	Search	**Planning**
States	Lisp data structures	Logical sentences
Actions	Lisp code	Preconditions/outcomes
Goal	Lisp code	Logical sentence (conjunction)
Plan	Sequence from s_0	Constraints on actions

Planning in Situation Calculus (I)

Planning as a logical inference problem (general purpose theorem proving), using situation calculus:

Initial state : a sentence about S_0

$At(Home, S_0) \wedge \neg Have(Milk, S_0) \wedge \neg Have(Bananas, S_0) \wedge \neg Have(Drill, S_0)$

Goal state : a logical query asking for "suitable situations"

$\exists\, s\ At(Home, s) \wedge Have(Milk, s) \wedge Have(Bananas, s) \wedge Have(Drill, s)$

Operators : action descriptions, such as a successor-state axiom

Remember: effect, frame, successor-state axiom

$$\begin{aligned}\forall a, s Have(Milk, Result(a,s)) \Leftrightarrow \quad & [(a = Buy(Milk) \wedge At(Supermarket, s) \\ & \vee (Have(Milk, s) \wedge (a \neq Drop(Milk))]\end{aligned}$$

Reminder, in situation calculus: $Result(action, s_i) = s_{i+1}$

Planning in Situation Calculus (II)

$\longrightarrow$ Situation calculus, logical inference is good, but has problems: complexity, semidecidable, quality of plans (e.g., loops) etc..

1. Restrict the language. Example: STRIPS
   Still based on logic, with a well-defined syntax, semantics, and proof theory.
2. Special-purpose algorithm: **planner**
   More efficient then general purpose theorem proving

Basic representations for Planning

STRIPS: expressive and 'efficient' [*argh*] language

1. Representations for states and goals
2. Representations for actions
3. Situation space and plan space
4. Representations for plans

STRIPS: States and goals are inputs

States: function-free <u>ground</u> literals
i.e., predicates applied to constant symbols, possibly negated

$$At(Home) \wedge \neg Have(Milk) \wedge \neg Have(Bananas) \wedge \neg Have(Drill) \ldots$$

$\longrightarrow$ **State description need not be complete**
Typically, planner keeps track of only small parts of states representation, keeps track of changes only

$\longrightarrow$ **"Negation-as-failure"-like principle**
if state description does not mention a given positive literal, then literal assumed to be false

Goals: conjunction of literals

$$At(Home) \wedge Have(Milk) \wedge Have(Bananas) \wedge Have(Drill)$$

may contain variables: $At(x) \wedge Sells(x, Milk)$

→ variables existentially quantified

Planner: asking for a sequence of actions to the goal
Theorem prover: asking whether the sentence is entailed by KB

STRIPS: actions (I)

Operator, 3 components: { action description, precondition, effect}

Action description: returned to the environment to act upon. In planner: a name.

Precondition: conjunction of <u>atoms</u> (*i.e.*, <u>positive</u> literals) must be true so that the operator can be applied

Effect: conjunction of literals (*i.e.*, positive or negative) describing change as a result of applying the operator (traditionally, `add-list` & `delete-list`)

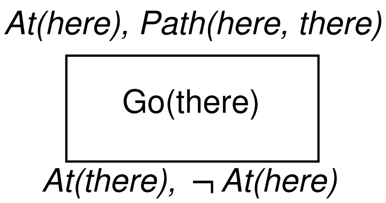
$$\begin{aligned} Op \quad & (Action : Go(there), \\ & Precondition : At(here) \wedge Path(here, there), \\ & Effect : At(there) \wedge \neg At(here)) \end{aligned}$$

→ Conditions on precondition and effect are relaxed in more advanced planners

STRIPS: actions(II)

$$
\begin{aligned}
Op \quad & (Action : Go(there), \\
& Precondition : At(here) \wedge Path(here, there), \\
& Effect : At(there) \wedge \neg At(here))
\end{aligned}
$$

Graphical notation:



Note: STRIPS does <u>not</u> have an explicit situation variable
(*e.g.*, s_i, before, and s_{i+1}, after)

STRIPS: actions(III)

Operator **schema** (family of operators): operator with variables

$$
\begin{aligned}
Op \quad & (Action : Go(x), \\
& Precondition : At(y) \wedge Path(y, x), \\
& Effect : At(x) \wedge \neg At(y))
\end{aligned}
$$

Usually, only fully instantiated operators are executable
$\rightarrow$ Planner has to make sure all variables are instantiated

STRIPS: actions(IV)

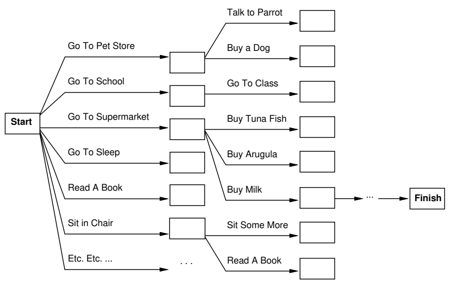
An operator o is applicable in a state s
if there is a way to instantiate variables in o
so that all literals in the precondition are true in s

$$Precond(o) \subseteq s$$

In $Result(o, s)$:
- — All positive literals in $\mathit{Effect}(o)$
- — All literals that held in s, except those that are negative in $\mathit{Effect}(o)$

Progression/regression planners: Situation space

Situation space planner: search in the space of situations of the world:



Generate a path in the situation space using any search algorithm
$\rightarrow$ yields a solution to the planning problem

Progression planner: high branching factor

Regression planner: Start from goal state (few literals) instead of initial state (too many literals)
possible: actions have all needed information
desirable: care only about relevant properties

Plan space

Planning search: search in the space of possible plans

Start with a partial plan, gradually move from incomplete/vague plans to complete, correct plans

Operators on partial plans:
- add a step
- impose an ordering between steps
- instantiate a previously unbound variable

$\surd$ **Refinement operators**: add constraints to a partial plan
gradually eliminate plans

$\times$ **Modification operators**: start with incorrect plans, then debug them

Partial order plans (I)

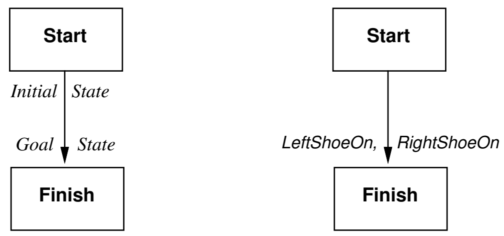
Example: put on your shoes

Actions: RightSock, RightShoe, LeftSock, LeftShoe

Initial plan: two steps, Start $\prec$ Finish

Start: Precondition: none
Effect: add the literals of initial state

Finish: Precondition: goal state
Effect: none

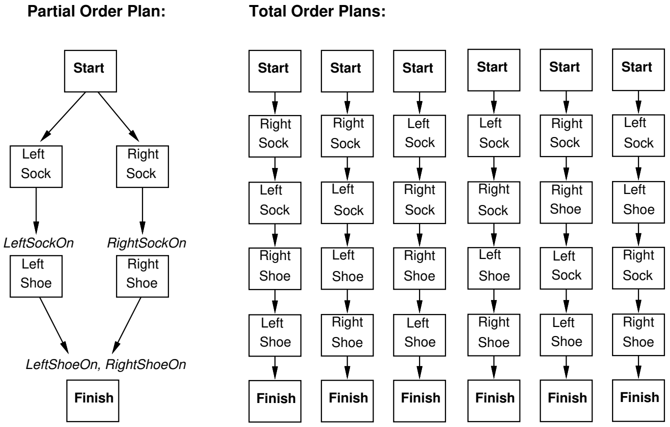


(a) (b)

Partial order plans (II)

Least commitment principle

Partial order plan, total order plan, linearization of a plan



A plan is complete iff every precondition is achieved

A precondition is achieved iff it is the effect of an earlier step and no possibly intervening step undoes it

Representations for plans

A plan is a data structure:

- A set of plan steps (a step is an operator)
- A set of step ordering constraints: *e.g.*, $S_i \prec S_j$
- A set of variable binding constraints: *e.g.*, $v = x$ where v is a variable and x is a constant.
- A set of **causal links**: record the purpose of a step
  A causal link $S_i \xrightarrow{c} S_j$: S_i achieves c for S_j

Plan (Steps: { $S_1 : Op(Action : Start)$,
$S_2 : Op(\ Action : Finish,$
$Precondition : RightShoeOn \wedge LeftShoeOn)\}$,
Orderings: $\{S_1 \prec S_2\}$,
Bindings: $\{\}$,
Links: $\{\}$)

(a) (b)

Representation of solutions

A solution is a plan the agent can execute $\begin{cases} \text{fully instantiated} \\ \text{totally ordered} \end{cases}$

Unsatisfactory:

- An ordering can be arbitrary, partial order is more natural
- Some agents can do things in parallel
- Need to retain flexibility for, later, combining plans

A solution is a **complete**, **consistent** plan

A plan is complete iff every precondition is achieved

A plan is consistent iff there is no contradictions
in the ordering ($S_i \prec S_j$ and $S_j \prec S_i$)
or binding constraints ($v = A, V = B$, $A \neq B$)

Partial-Order Planning: Example (I)

PO-planner that:

- searches through the plan space
- starts with an initial plan (initial state + goal state)
- at each iteration, adds one more step
- remains focused: only considers adding steps that serve to achieve a precondition that has not yet been achieved
  $\rightarrow$ causal links allow keeping track of this

Problem: milk-banana-drill simple problem

Assumptions:

- *Go* action used to travel between any two locations
- *Buy* action ignores money
- *HWS* = hardware store, *SM* = supermarket

PO Planning: Example (II)

- **Initial state:**

$Op\,(Action : Start,$
$\quad Effect : At(Home) \wedge Sells(HWS, Drill)$
$\qquad \wedge Sells(SM, Milk)$
$\qquad \wedge Sells(SM, Banana))$

- **Goal state:**

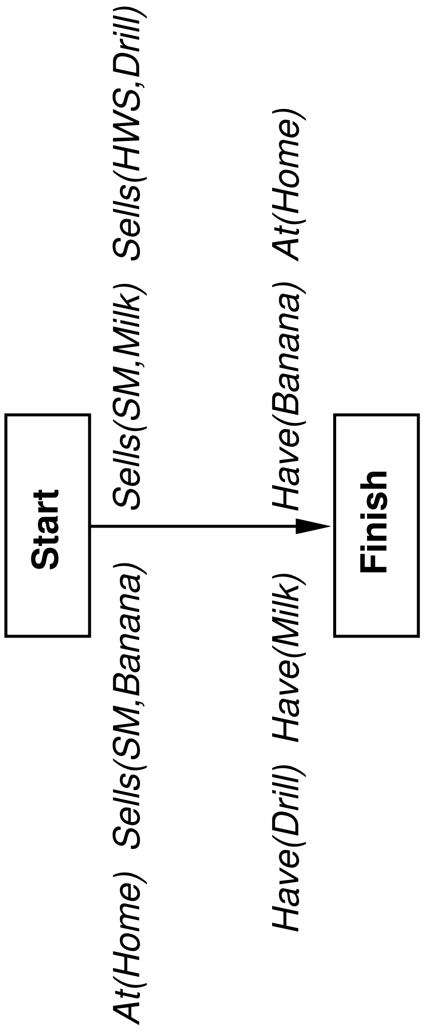
$Op\,(Action : Finish,$
$\quad Precond : Have(Drill) \wedge Have(Bananas)$
$\qquad \wedge Have(Milk) \wedge At(Home))$

- **Actions:**

$Op\,(Action : Go(there),$
$\quad Precond : At(here),$
$\quad Effect : At(there) \wedge \neg At(here))$

$Op\,(Action : Buy(x),$
$\quad Precond : At(store) \wedge Sells(store, x),$
$\quad Effect : Have(x))$

Initial plan:

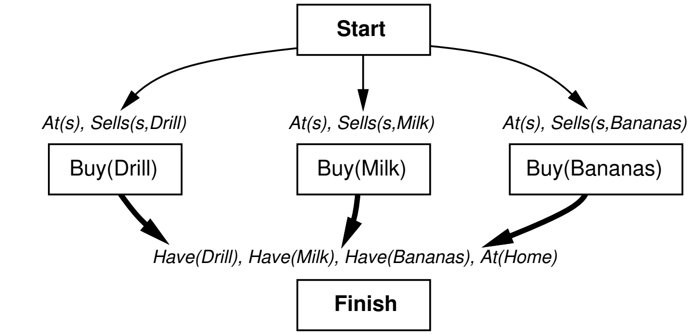


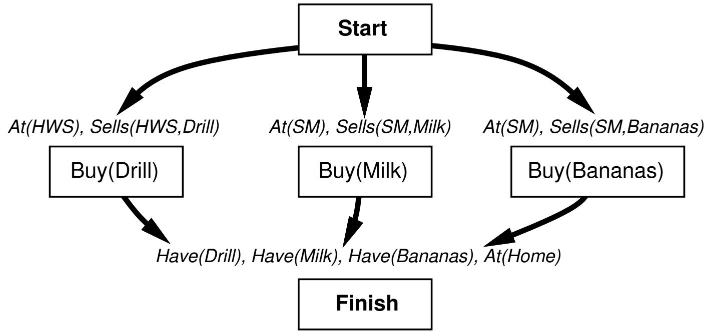
→ choose actions that achieve *Precond* of goal state

→ In general, there are correct and incorrect choices of actions.

→ In this case only one action: *Buy*

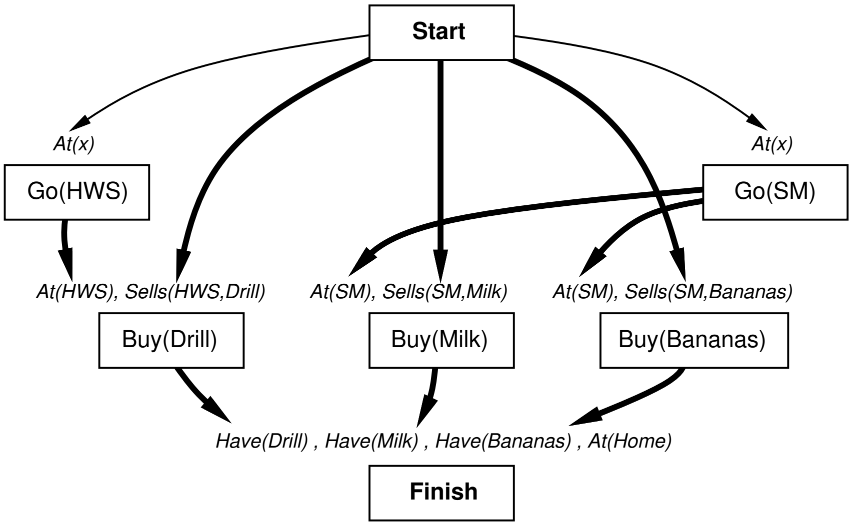
⟶ Bold arrow denotes a causal link
causal linked are <u>protected</u>: no step
that deletes $Have(Drill)$ may be inserted
between $Buy(Drill)$ and $Finish$





→ No action to achieve $Sell$, achieved by linking them to $Start$

$\longrightarrow$ Action to achieve $At(x)$ is $Go(x)$



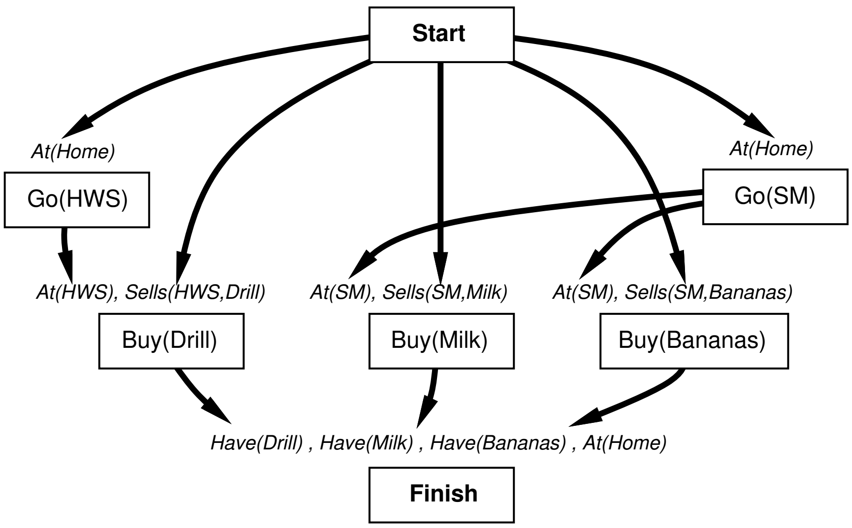
$\longrightarrow$ Try to achieve $At(x)$ with by linking to $Start$

$\longrightarrow$ Problem:

$Go(HWS)$ adds $At(HWS)$ and deletes $At(home)$

$Go(SM)$ is no longer feasible

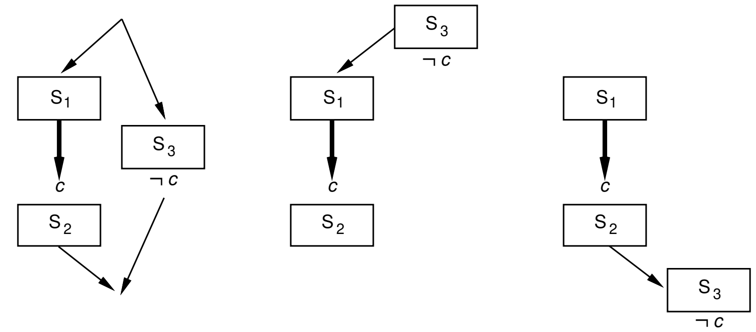
$\longrightarrow$ Solution: POP protects causal links



Protection of causal links:

Threats: steps that might delete (clobber) protected condition

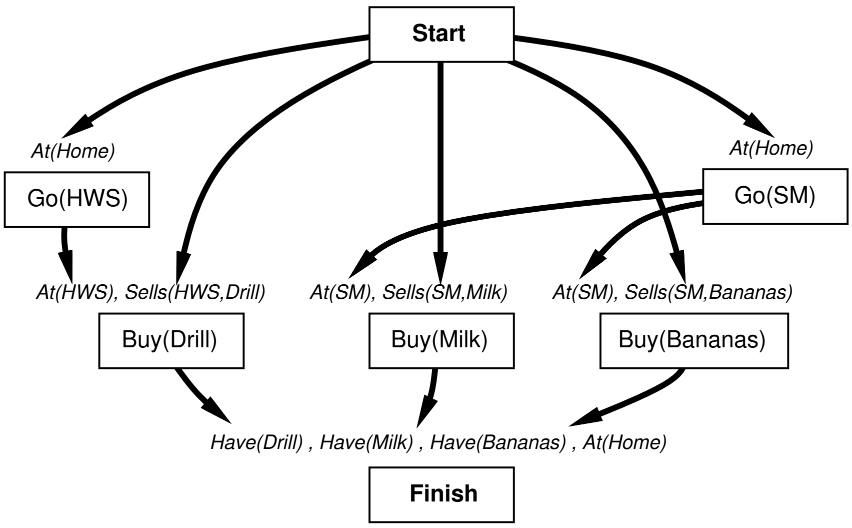
Solved by adding ordering constraints: $\begin{cases} \text{demotion } (S_3 \text{ placed before}) \\ \text{promotion } (S_3 \text{ placed before}) \end{cases}$



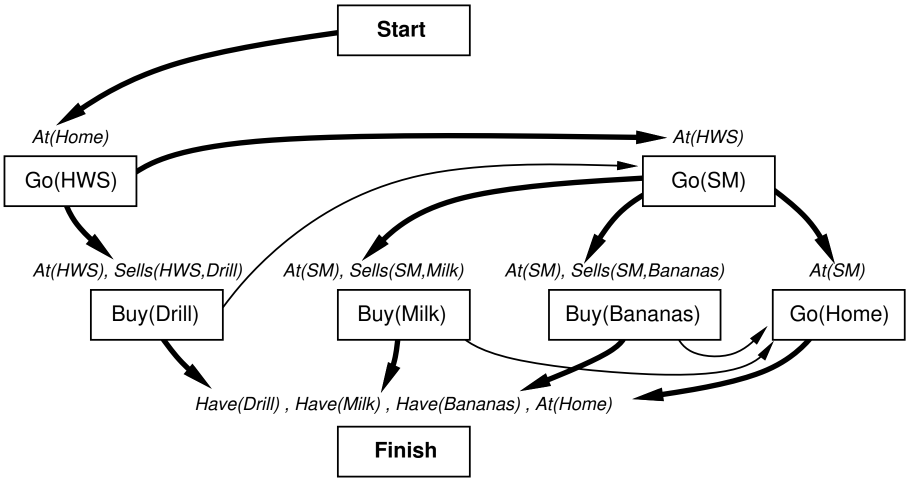
(a) (b) (c)

⟶ Promotion? Demotion?... none works?
backup to a different choice point in the plan

⟶ Try to achieve $At(x)$ of $Go(SM)$
by adding causal link from $Go(HWS)$ to $Go(SM)$

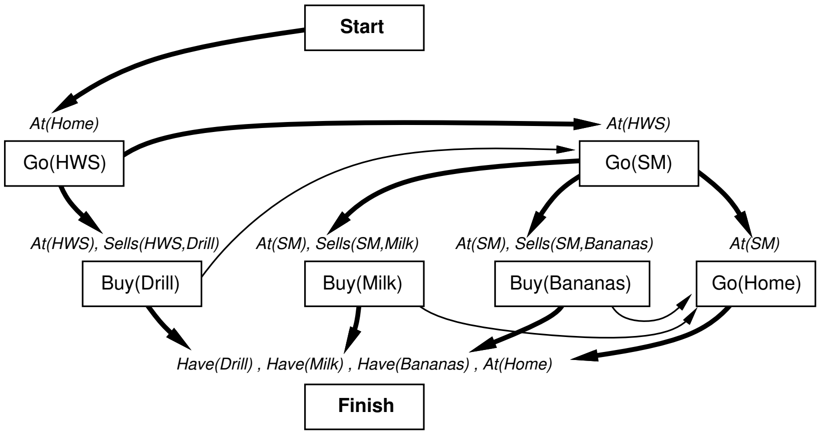


$\longrightarrow$ $Go(SM)$ threatens $At(HWS)$, precond for $Buy(Drill)$
Promote $Go(SM)$

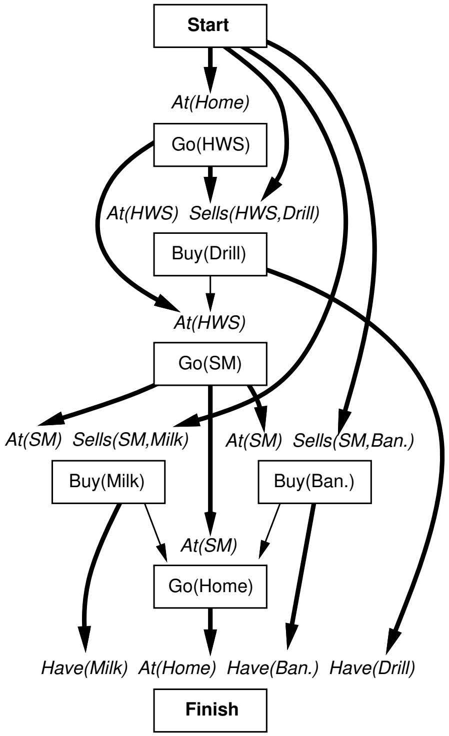


$\longrightarrow$ All precond achieved except $At(Home)$ of $Finish$
Add step $Go(home)$

- link to $Start$? No way to resolve threats $Go(SM)$, $Go(HWS)$
- link to $At(x)$ to $Go(HWS)$? No way to resolve threat $Go(SM)$
- link to $At(x)$ to $Go(SM)$? threat to $Buy(Milk)$ and $Buy(Bananas)$ solved by promoting $Go(home)$



Finally...
nearly a totally ordered plan (except for buy bananas and buy milk)



POPlanner

- solves in few steps what search solves in many many steps..
- least commitment: search is required only when subplans interact
- causal links recognize when to abandon a doomed plan without wasting time expanding irrelevant parts of the plan..

POPlanner algorithm

function POP(*initial, goal, operators*) **returns** *plan*

plan ← MAKE-MINIMAL-PLAN(*initial, goal*)
loop do
  if SOLUTION?(*plan*) **then return** *plan*
  S_{need}, c ← SELECT-SUBGOAL(*plan*)
  CHOOSE-OPERATOR(*plan, operators*, S_{need}, c)
  RESOLVE-THREATS(*plan*)
end

---

function SELECT-SUBGOAL(*plan*) **returns** S_{need}, c

pick a plan step S_{need} from STEPS(*plan*)
  with a precondition c that has not been achieved
return S_{need}, c

---

procedure CHOOSE-OPERATOR(*plan, operators*, S_{need}, c)

choose a step S_{add} from *operators* or STEPS(*plan*) that has c as an effect
if there is no such step **then fail**
add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS(*plan*)
add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS(*plan*)
if S_{add} is a newly added step from *operators* **then**
  add S_{add} to STEPS(*plan*)
  add $Start \prec S_{add} \prec Finish$ to ORDERINGS(*plan*)

---

procedure RESOLVE-THREATS(*plan*)

for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS(*plan*) **do**
  choose either
    Promotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)
    Demotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)
  if not CONSISTENT(*plan*) **then fail**
end

→ Backtrack: when fail to resolve a threat or find operator

→ Select-subgoal is <u>not</u> subject to backtracking

POPlanner algorithm

- is a regression planner
- is sound, complete, and <u>systematic</u> (no repetition)
- ...
- go back to shopping example and apply mechanically