

PLANNING

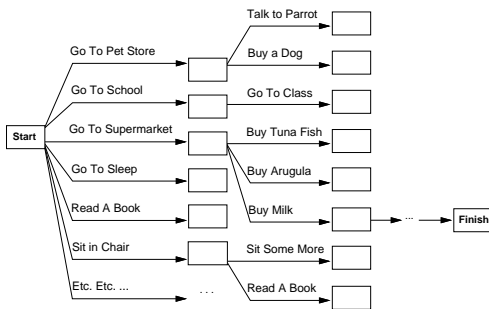
CHAPTER 11

Outline

- ◇ Search vs. planning
- ◇ STRIPS operators
- ◇ Partial-order planning

Search vs. planning

Consider the task *get milk, bananas, and a cordless drill*
 Standard search algorithms seem to fail miserably:



After-the-fact heuristic/goal test inadequate

Search vs. planning contd.

Planning systems do the following:

- 1) open up action and goal representation to allow selection
- 2) divide-and-conquer by subgoaling
- 3) relax requirement for sequential construction of solutions

| | 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 |

STRIPS operators

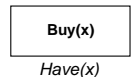
Tidily arranged actions descriptions, restricted language

ACTION: $Buy(x)$

PRECONDITION: $At(p), Sells(p, x)$

EFFECT: $Have(x)$

$At(p) Sells(p, x)$



[Note: this abstracts away many important details!]

Restricted language \Rightarrow efficient algorithm

Precondition: conjunction of positive literals

Effect: conjunction of literals

A complete set of STRIPS operators can be translated into a set of successor-state axioms

Partially ordered plans

Partially ordered collection of steps with

Start step has the initial state description as its effect

Finish step has the goal description as its precondition

causal links from outcome of one step to precondition of another

temporal ordering between pairs of steps

Open condition = precondition of a step not yet causally linked

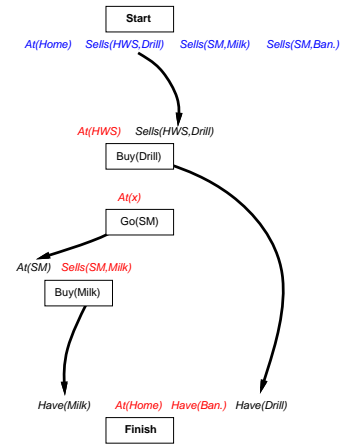
A plan is *complete* iff every precondition is achieved

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

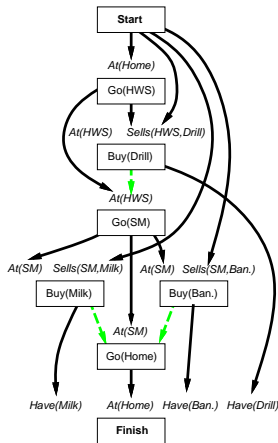
Example



Example



Example



Planning process

Operators on partial plans:

- add a link from an existing action to an open condition
- add a step to fulfill an open condition
- order one step wrt another to remove possible conflicts

Gradually move from incomplete/vague plans to complete, correct plans

Backtrack if an open condition is unachievable or if a conflict is unresolvable

POP algorithm sketch

```

function POP(initial, goal, operators) returns plan
  plan ← MAKE-MINIMAL-PLAN(initial, goal)
  loop do
    if SOLUTION?(plan) then return plan
    Sneed, c ← SELECT-SUBGOAL(plan)
    CHOOSE-OPERATOR(plan, operators, Sneed, c)
    RESOLVE-THREATS(plan)
  end
  
```

```

function SELECT-SUBGOAL(plan) returns Sneed, c
  pick a plan step Sneed from STEPS(plan)
  with a precondition c that has not been achieved
  return Sneed, c
  
```

POP algorithm contd.

```

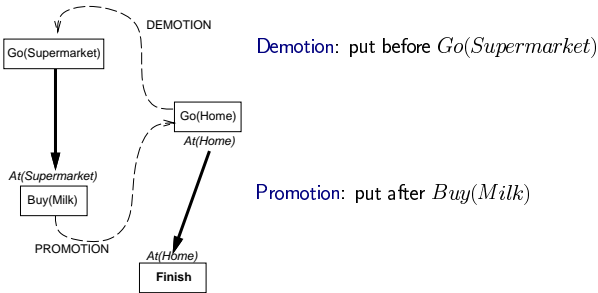
procedure CHOOSE-OPERATOR(plan, operators, Sneed, c)
  choose a step Sadd from operators or STEPS(plan) that has c as an effect
  if there is no such step then fail
  add the causal link Sadd → Sneed to LINKS(plan)
  add the ordering constraint Sadd < Sneed to ORDERINGS(plan)
  if Sadd is a newly added step from operators then
    add Sadd to STEPS(plan)
    add Start < Sadd < Finish to ORDERINGS(plan)
  
```

```

procedure RESOLVE-THREATS(plan)
  for each Sthreat that threatens a link Si → Sj in LINKS(plan) do
    choose either
      Demotion: Add Sthreat < Si to ORDERINGS(plan)
      Promotion: Add Sj < Sthreat to ORDERINGS(plan)
    if not CONSISTENT(plan) then fail
  end
  
```

Clobbering and promotion/demotion

A **clobberer** is a potentially intervening step that destroys the condition achieved by a causal link. E.g., $Go(Home)$ clobbers $At(Supermarket)$:



Properties of POP

Nondeterministic algorithm: backtracks at **choice** points on failure:

- choice of S_{add} to achieve S_{need}
- choice of demotion or promotion for clobberer
- selection of S_{need} is irrevocable

POP is sound, complete, and **systematic** (no repetition)

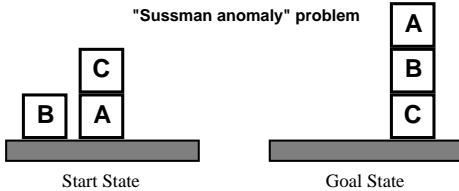
Extensions for disjunction, universals, negation, conditionals

Can be made efficient with good heuristics derived from problem description

Particularly good for problems with many loosely related subgoals

Example: Blocks world

"Sussman anomaly" problem



$Clear(x) \ On(x,z) \ Clear(y)$

PutOn(x,y)

$\sim On(x,z) \ \sim Clear(y)$
 $Clear(z) \ On(x,y)$

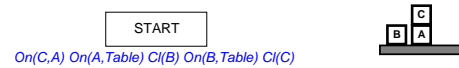
+ several inequality constraints

$Clear(x) \ On(x,z)$

PutOnTable(x)

$\sim On(x,z) \ Clear(z) \ On(x, Table)$

Example contd.

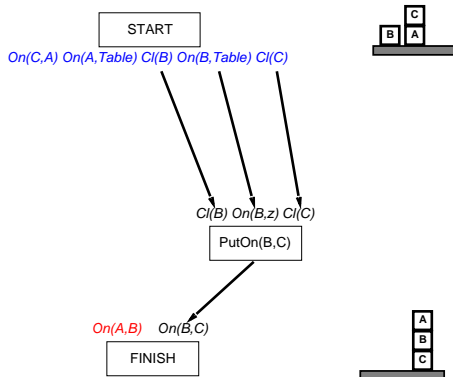


$On(A,B) \ On(B,C)$

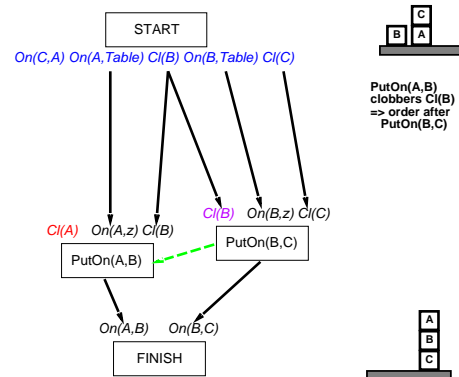
FINISH



Example contd.



Example contd.



Example contd.

