Planning

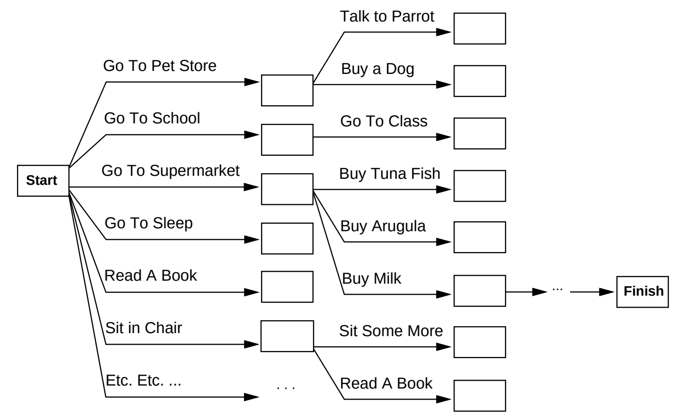
Chapter 10

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.

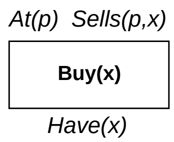
Planning systems do the following:
1) open up action and goal representation to allow selection
2) divide-and-conquer by subgoaling
3) relax requirement for sequential construction of solutions

	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: $Have(x)$



[Note: this abstracts away many important details!]

Restricted language $\Rightarrow$ efficient algorithm

- Precondition: conjunction of positive literals
- Effect: conjunction of literals
  - postive effect: add literals
  - negative effect: remove literals (negated literals)

Forward State-space Search

◇ aka Progression Planning

◇ similar to Forward Chaining

◇ State-space formulation

- Initial State: initial KB
- Actions: operators whose preconditions are satisfied
  - successors:
    - postive effect: add literals
    - negative effect: remove literals
- Goal test: goal state
- Step cost: typically 1

Backward State-space Search

◇ aka Regression Planning

◇ similar to Backward Chaining

◇ Difficult if the goal is described as constraints (e.g. 4 gallons in the large jug)—potentially many goal states.

◇ A goal can be divided into sub-goals (children).

◇ State-space formulation

- Initial State: goal state
- Actions: operations that can acheive the goal/sub-goal
  - not undo any super-goals [parent goals/preconditions]
  - successors:
    - sub-goals (unsatisfied precoditions)
- Goal test: no sub-goals (no unsatisfied preconditions)

Admissible Heuristics

◇ Relaxed problem

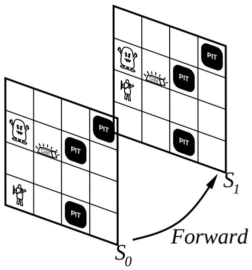
- remove all precoditions—every action is applicable
- remove all negative effects—no action removes a literal
  (note that the goal is a conjuction of literals)
- subgoal independence—achieving one subgoal does not affect achieving another subgoal

Keeping track of change–Situation Calculus

Facts hold in situations, rather than eternally
E.g., $Holding(Gold, Now)$ rather than just $Holding(Gold)$

Situation calculus is one way to represent change in FOL:
Adds a situation argument to each non-eternal predicate
E.g., Now in $Holding(Gold, Now)$ denotes a situation

Situations are connected by the $Result$ function
$Result(a, s)$ is the situation that results from doing a in s



Describing actions I

"Effect" axiom—describe changes due to action
$\forall s \;\; AtGold(s) \;\Rightarrow\; Holding(Gold, Result(Grab, s))$

"Frame" axiom—describe **non-changes** due to action
$\forall s \;\; HaveArrow(s) \;\Rightarrow\; HaveArrow(Result(Grab, s))$

Frame problem: find an elegant way to handle non-change
(a) representation—avoid frame axioms
(b) inference—avoid repeated "copy-overs" to keep track of state

Qualification problem: true descriptions of real actions require endless caveats—what if gold is slippery or nailed down or . . .

Ramification problem: real actions have many secondary consequences—what about the dust on the gold, wear and tear on gloves, . . .

Describing actions II

Successor-state axioms solve the representational frame problem

Each axiom is "about" a **predicate** (not an action per se):

P true afterwards $\Leftrightarrow$ [an action made P true
$\vee$ P true already and no action made P false]

For holding the gold:
$\forall a, s \;\; Holding(Gold, Result(a, s)) \;\Leftrightarrow$
$[(a = Grab \wedge AtGold(s))$
$\vee\, (Holding(Gold, s) \wedge a \neq Release)]$

Making Plans

Initial condition in KB:
$At(Agent, [1, 1], S_0)$
$At(Gold, [1, 2], S_0)$

Query: $Ask(KB, \exists s \;\; Holding(Gold, s))$
i.e., in what situation will I be holding the gold?

Answer: $\{s/Result(Grab, Result(Forward, S_0))\}$
i.e., go forward and then grab the gold

This assumes that the agent is interested in plans starting at S_0 and that S_0 is the only situation described in the KB

Making plans: A better way

Represent plans as action sequences $[a_1, a_2, \ldots, a_n]$

$PlanResult(p, s)$ is the result of executing p in s

Then the query $Ask(KB, \exists\, p\;\; Holding(Gold, PlanResult(p, S_0)))$
has the solution $\{p/[Forward, Grab]\}$

Definition of $PlanResult$ in terms of $Result$:

$\forall\, s\;\; PlanResult([\,], s) = s$

$\forall\, a, p, s\;\; PlanResult([a|p], s) = PlanResult(p, Result(a, s))$

Planning systems are special-purpose reasoners designed to do this type of inference more efficiently than a general-purpose reasoner

Partial Order Planning

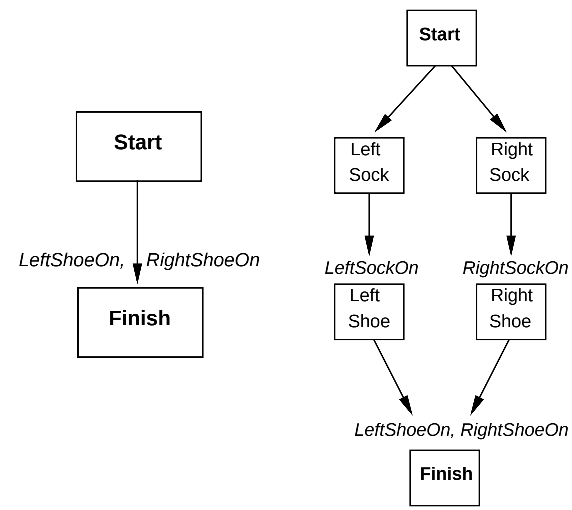
◇ sequential planning: forward (or backward) step-by-step search

◇ Consider planning a trip to New York by flying

1. start with finding how to get from home to the Melboune airport
2. start with finding how to get from the New York airport to hotel
3. start with finding a plane ticket from Melbourne to New York

◇ least commitment strategy—delay making committments to steps that are less important/constrained

Partial Order Planning

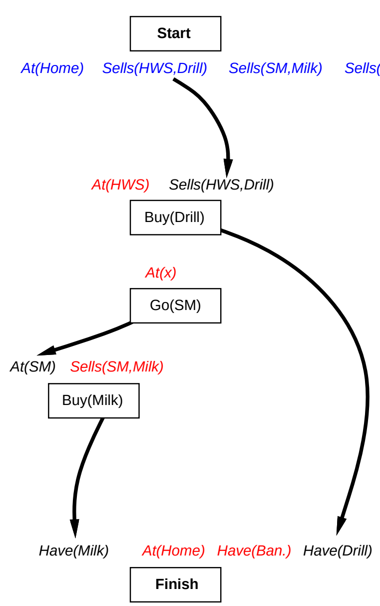


Components of Partial Order Planning

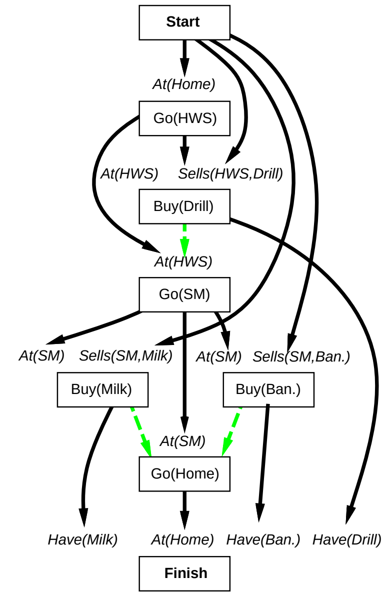
- Actions
  - "Start" action: no preconditions, effects = initial state
  - "Finish" action: preconditions = goal state, no effects
  - (regular) actions with precondtions and effects
- Ordering contraints between actions
  - $A \prec B$: A is before B (partial order)
  - $LeftSock \prec LeftShoe$
- Causal links from effect of one action to the precondition of another
  - $A \xrightarrow{c} B$: A acheives precondition c for B
  - $LeftSock \xrightarrow{LeftSockOn} LeftShoe$
  - other actions cannot conflict with the causal link: $\neg LeftSockOn$
- Open preconditions
  - not acheived by any action yet
  - planner: add actions until there are no open preconditions

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

Topological Sorting in graphs

POP algorithm sketch

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

POP algorithm contd.

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*)

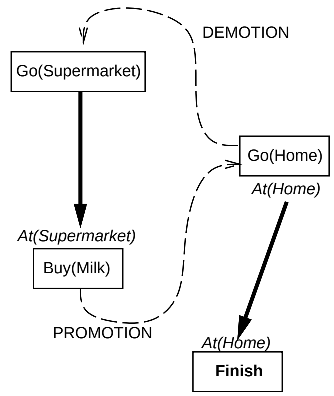
---

procedure Resolve-Threats(*plan*)

for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in Links(*plan*) **do**
  choose either
    Demotion: Add $S_{threat} \prec S_i$ to Orderings(*plan*)
    Promotion: Add $S_j \prec S_{threat}$ 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 $At(Supermarket)$:



Demotion: put before $Go(Supermarket)$

Promotion: put after $Buy(Milk)$

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

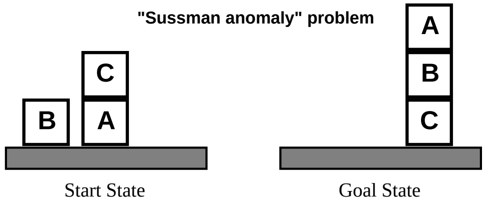
POP is sound, complete, and systematic (no repetition)

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



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

PutOn(x,y)

~On(x,z) ~Clear(y)
Clear(z) On(x,y)

Clear(x) On(x,z)

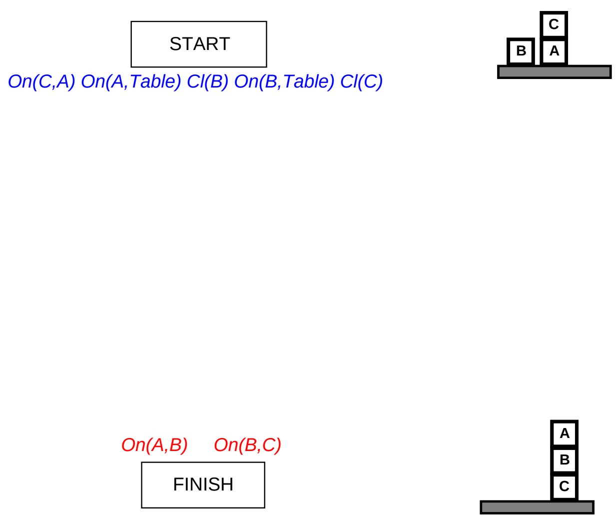
PutOnTable(x)

~On(x,z) Clear(z) On(x,Table)

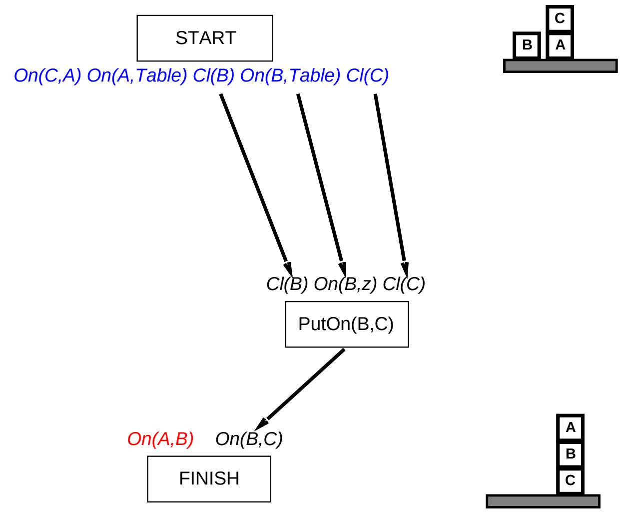
+ several inequality constraints

[Linear planners (find a plan for each subgoal and concatenate the plans) can't find a solution]

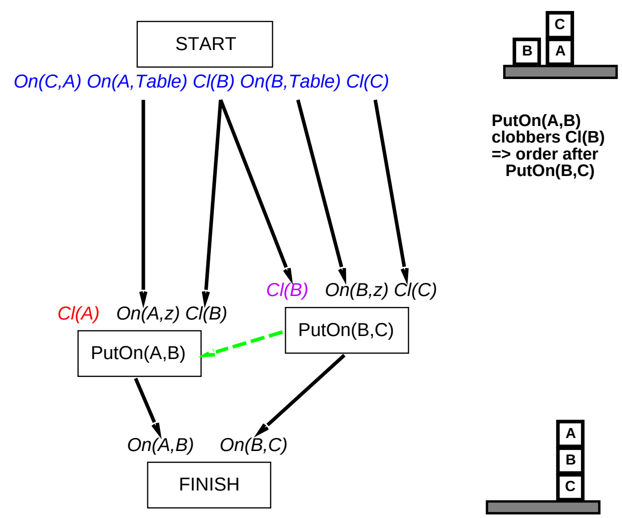
Example contd.



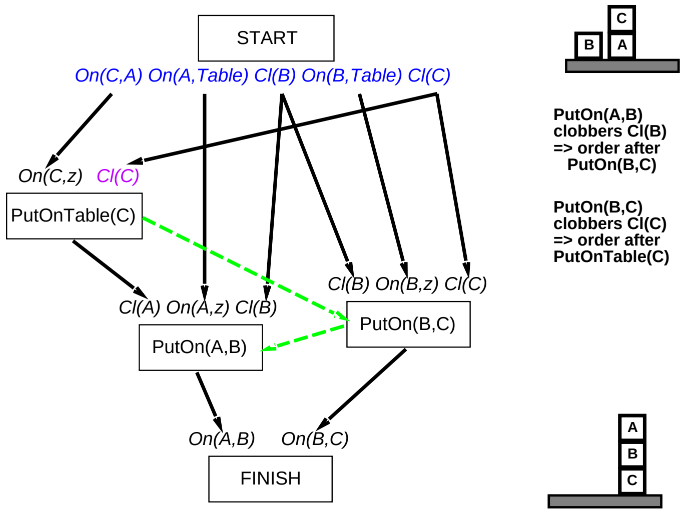
Example contd.



Example contd.



Example contd.



Heuristics

◇ Which open precondition to choose?

- most constrained open precondition
  - can be satisfied in the fewest number of ways
- can provide substantial speedups
  - if it can't be satisfied, stop early and return fail
  - if it can be satisfied by only one way, no choice anyhow and can reduce the number of possibilities later on