

Planning

CHAPTER 11

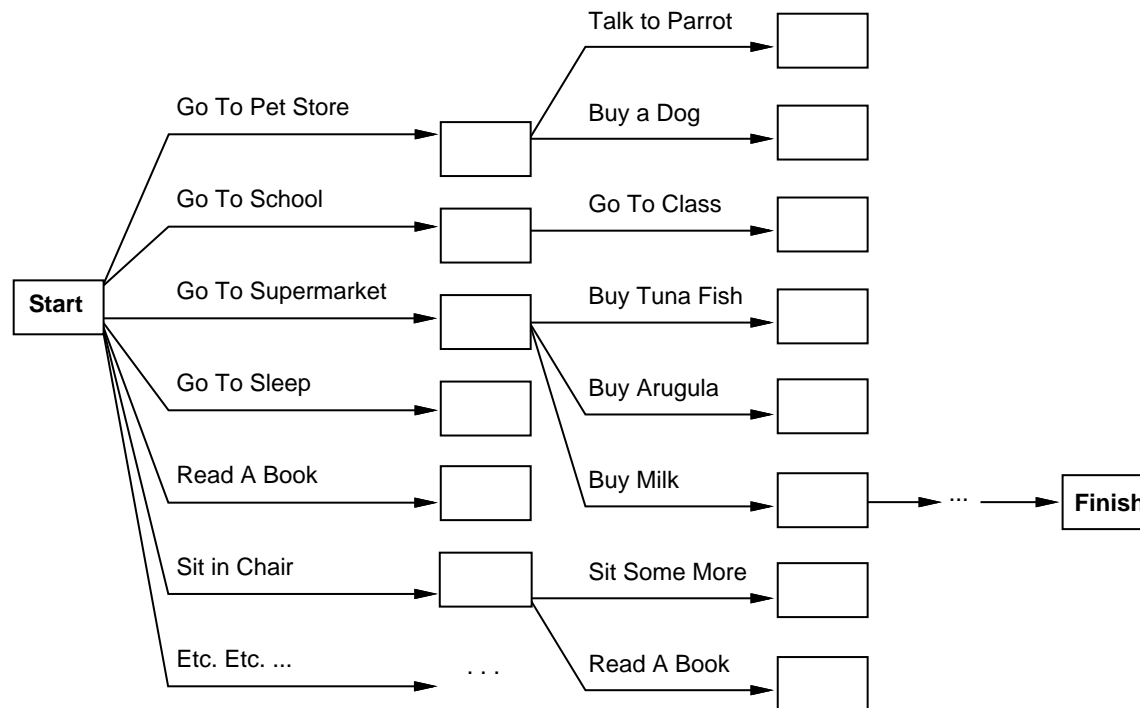
Outline

- ◇ 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

Planning in situation calculus

$PlanResult(p, s)$ is the situation resulting from executing p in s

$$PlanResult([], s) = s$$

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

Initial state $At(Home, S_0) \wedge \neg Have(Milk, S_0) \wedge \dots$

Actions as Successor State axioms

$$Have(Milk, Result(a, s)) \Leftrightarrow$$

$$[(a = Buy(Milk) \wedge At(Supermarket, s)) \vee (Have(Milk, s) \wedge a \neq \dots)]$$

Query

$$s = PlanResult(p, S_0) \wedge At(Home, s) \wedge Have(Milk, s) \wedge \dots$$

Solution

$$p = [Go(Supermarket), Buy(Milk), Buy(Bananas), Go(HWS), \dots]$$

Principal difficulty: unconstrained branching, hard to apply heuristics

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

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

Buy(x)

$Have(x)$

State space vs. plan space

Standard search: node = concrete world state

Planning search: node = partial plan

Defn: open condition is a precondition of a step not yet fulfilled

Operators on partial plans:

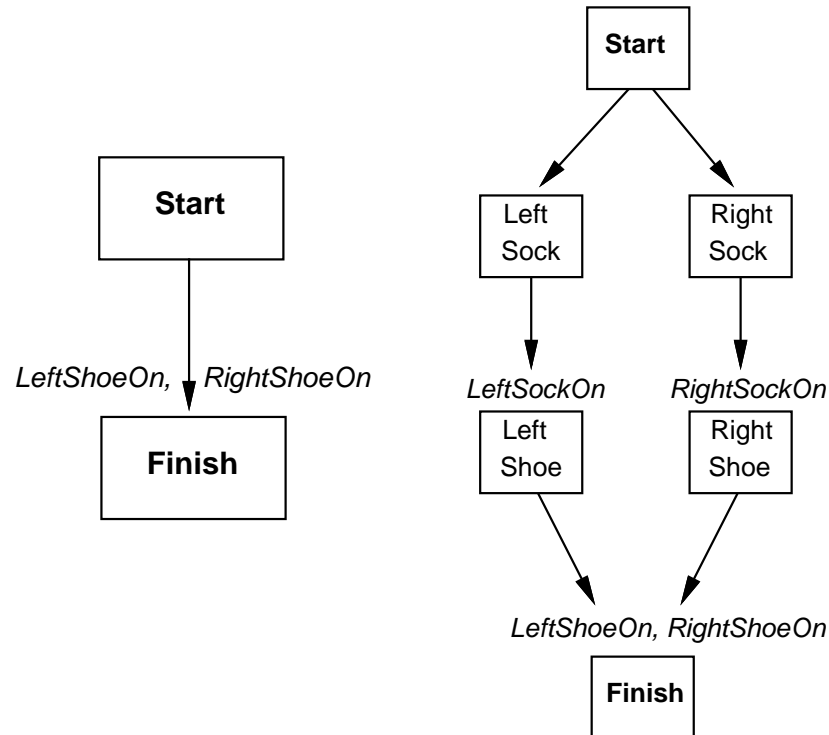
add a link from an existing action to an open condition

add a step to fulfill an open condition

order one step wrt another

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

Partially ordered plans



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

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**

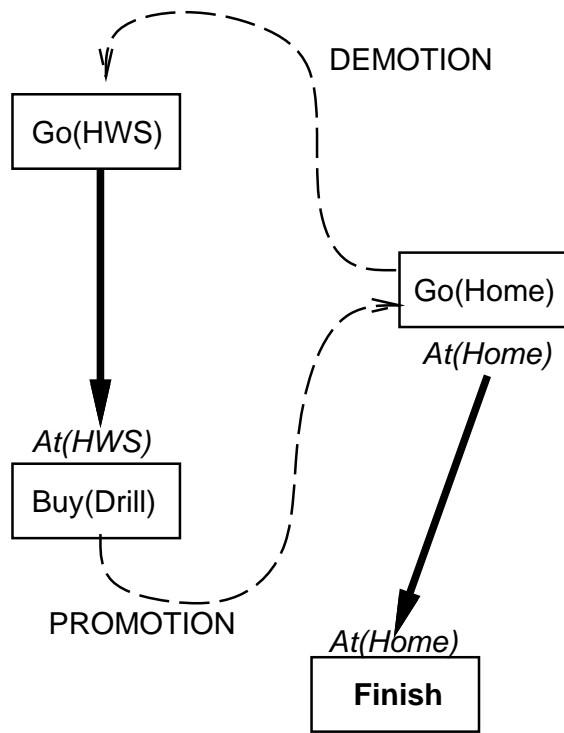
end

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

Extensions for disjunction, universals, negation, conditionals

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 $At(HWS)$:

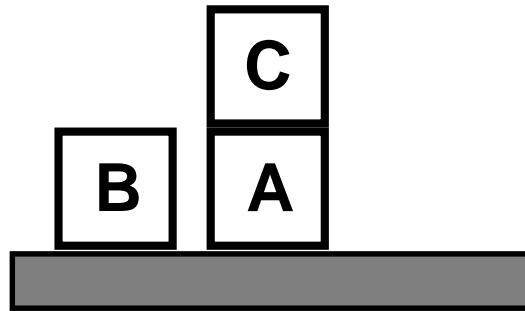


Demotion: put before $Go(HWS)$

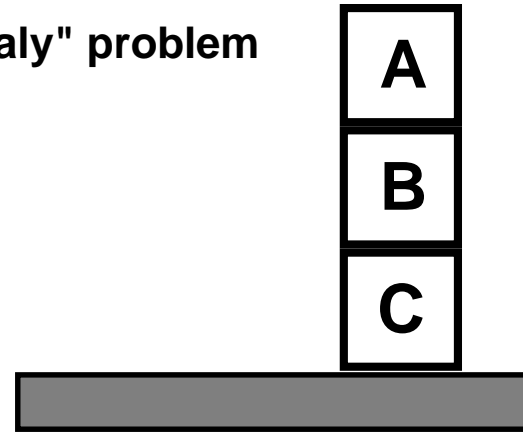
Promotion: put after $Buy(Drill)$

Example: Blocks world

"Sussman anomaly" problem



Start State



Goal State

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

PutOn(x,y)

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

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

PutOnTable(x)

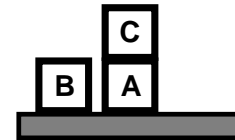
$\sim On(x,z) \ Clear(z) \ On(x, Table)$

+ several inequality constraints

Example contd.

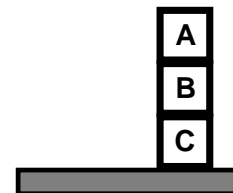
START

$On(C,A)$ $On(A,Table)$ $Cl(B)$ $On(B,Table)$ $Cl(C)$

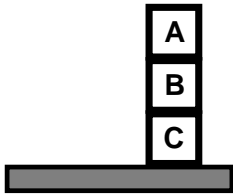
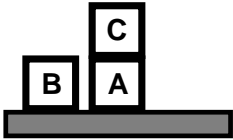
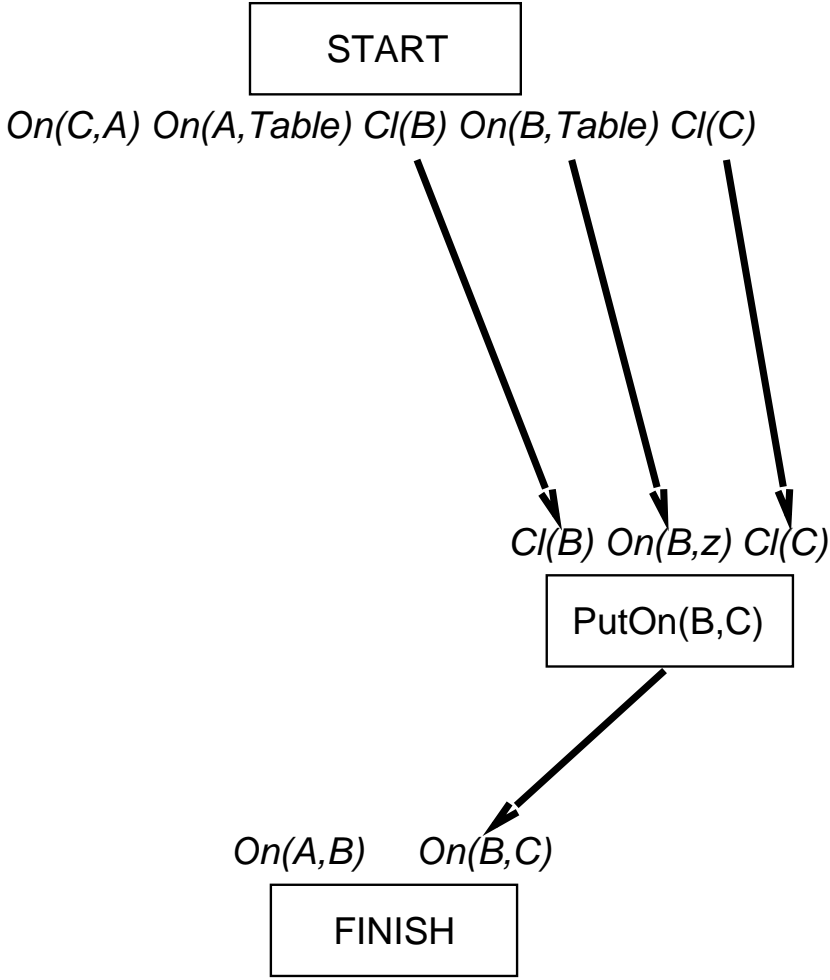


$On(A,B)$ $On(B,C)$

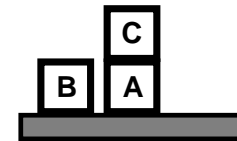
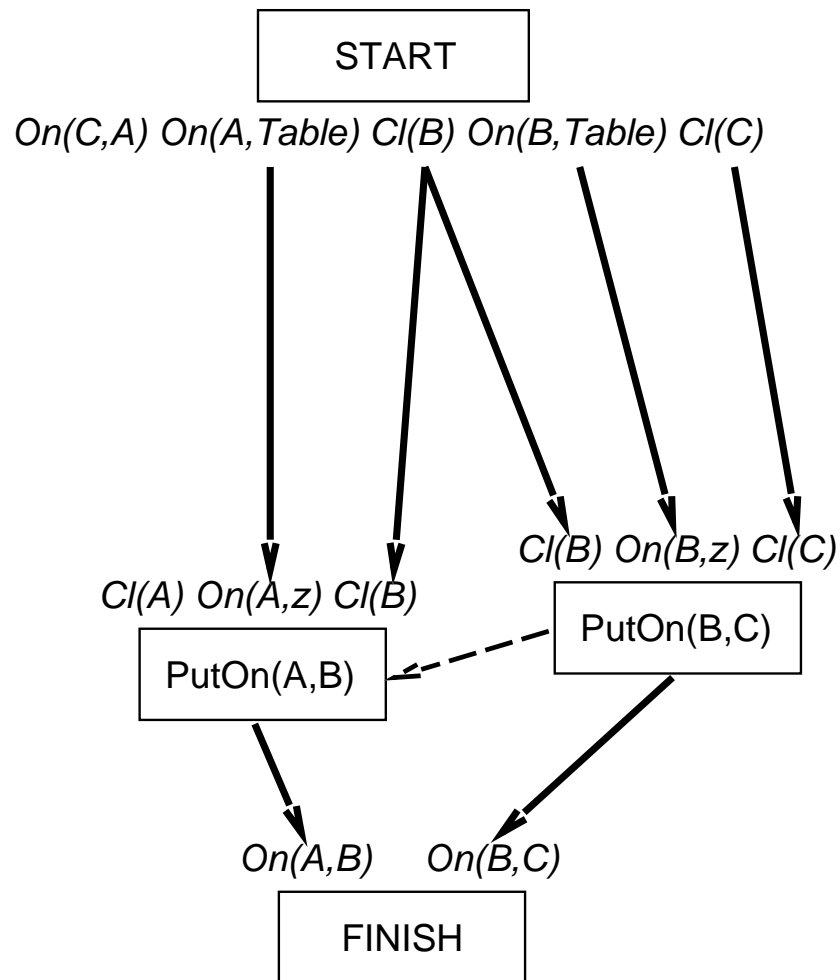
FINISH



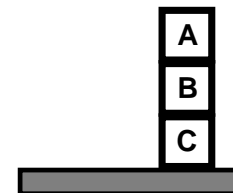
Example contd.



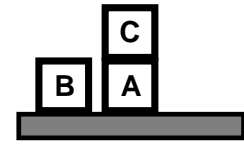
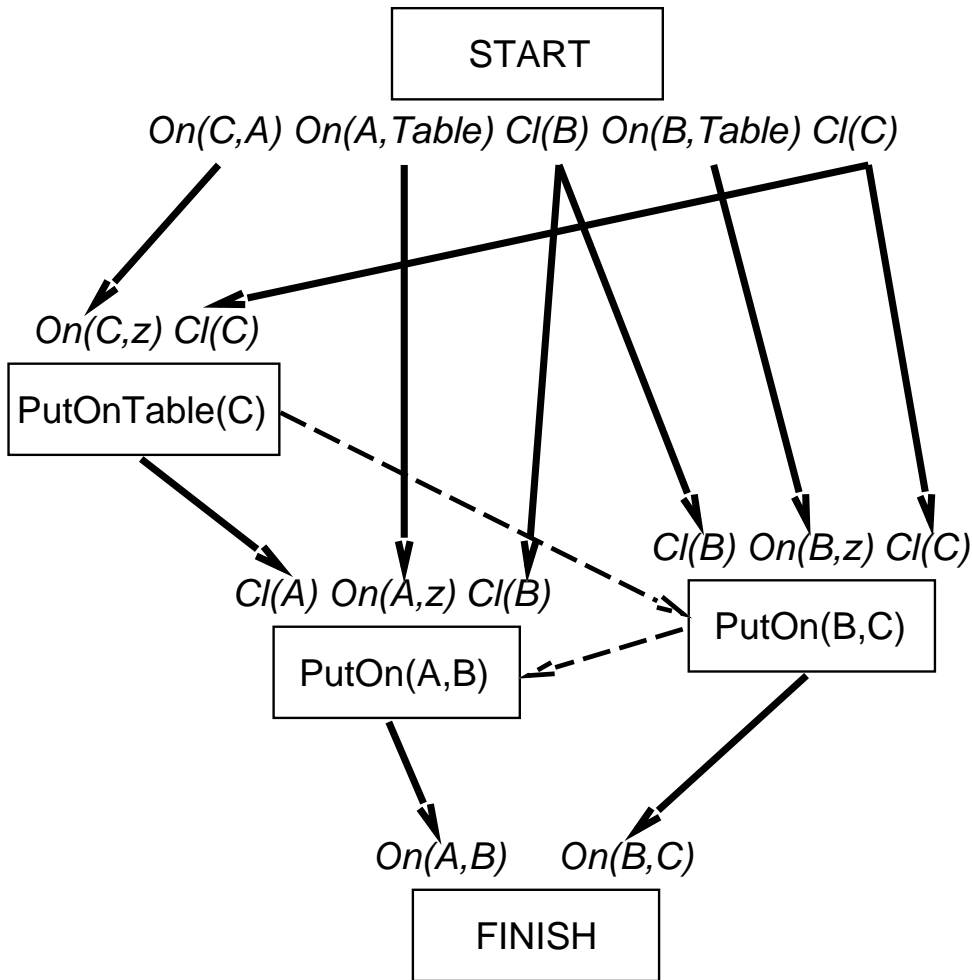
Example contd.



PutOn(A,B)
 clobbers Cl(B)
 => order after
 PutOn(B,C)



Example contd.



PutOn(A,B)
 clobbers Cl(B)
 => order after
 PutOn(B,C)

PutOn(B,C)
 clobbers Cl(C)
 => order after
 PutOnTable(C)

