Artificial Intelligence II

2013/2014 - Prof: Daniele Nardi, Joachim Hertzberg

Exercitation 3 - Roberto Capobianco

Planning: STRIPS, Partial Order Plans, Planning Graphs

STRIPS (Recap-1)

- Start situation;
- Goal conditions;
- Operator schemata
  - Preconditions;
  - Postconditions;
- Plan: pair $\langle O, < \rangle$ of a set O of operator instances and an ordering relation $<$ on O.
- Operator: pair $\langle R, S \rangle$ of sets R, S of ground facts.
  - R: operator's preconditions (what must be true to execute it?);
  - S: postconditions (what are the effects of its execution?);
- Progression or Regression:
  - Progression: search forward from initial state to goal;
  - Regression: search backward from goal to initial state.

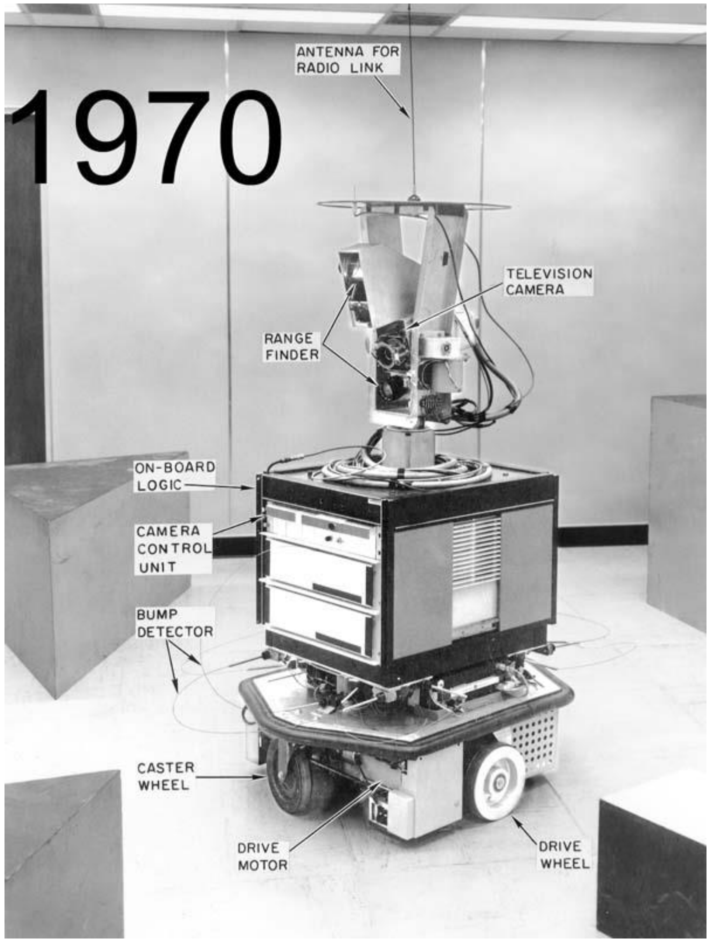
STRIPS (Recap-2)

- STRIPS Algorithm, with Regression:

```
function STRIPS-PLANNER(⟨$S$,$O$,$F$⟩) returns a plan (operator sequence)
input: ⟨$S$,$O$,$F$⟩, a STRIPS problem representation
local variables: $P$, $P_o$ : Plan (operator sequence)
begin
1.   $P \leftarrow \lambda$ (empty sequence)
2.   while still goals of $F$ open in $S$ do
       2.1  choose goal $f \in F$ open in $S$
       2.2  choose operator instance $o \in O$ adding $f$ ;
            return(failure) if no such $o$ exists
       2.3  $P_o \leftarrow$ STRIPS-PLANNER(⟨$S$,$O$, $o.preconditions$⟩) $\oplus$ $o$;
            if  $P_o$=fail  then  return(failure)                ($\oplus$ = concatenation)
       2.4  $S \leftarrow P_o(S)$;   $P \leftarrow P \oplus P_o$
3.   return($P$) end
```

STRIPS & Shakey

- Stanford Robot controlled by STRIPS;
- At the time planner could find plans beyond robots abilities;
- Actions:
  - Move;
  - Pushing movable objects;
  - Climb onto and down from objects;
  - Turn light switches on and off.

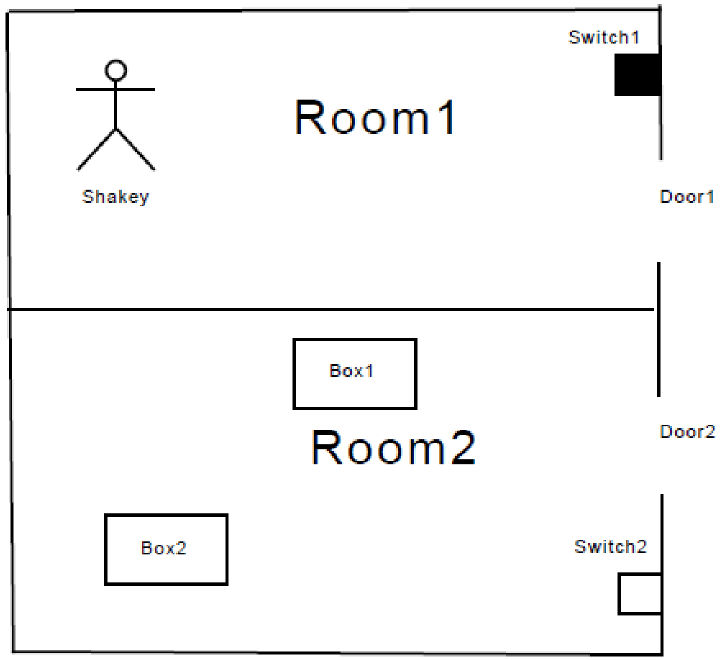


Shakey Problem (1)

- Describe Shakey's Actions in STRIPS:
  - Go(x, y) where x and y are locations in the same room (there is a door between the rooms);
  - Push(b, x, y) to push box b from x to y (both locations in the same room);
  - ClimbUp(b, x) and ClimbDown(b, x)
  - TurnOn(s, x) and TurnOff(s, x);

Shakey Problem (2)

- Describe:
  - The initial state in figure;
  - The goal state where Box2 is in Shakey's initial position;
- Construct a plan using Progression and one using Regression.



Shakey's Actions (1)

- Action(

  GoTo(x, y),

  PRECOND : $At(Shakey, x) \wedge x \neq y \wedge sameRoom(x, y)$

  EFFECT : $\neg At(Shakey, x) \wedge At(Shakey, y)$

)

- Action(

  Push(b, x, y),

  PRECOND : $Box(b) \wedge At(b, x) \wedge x \neq y \wedge sameRoom(x, y) \wedge At(Shakey, x)$

  EFFECT : $\neg At(b, x) \wedge At(b, y)) \wedge \neg At(Shakey, x) \wedge At(Shakey, y)$

)

Shakey's Actions (2)

- Action(

  ClimbUp(b, x),

  PRECOND : Box(b) ∧ At(Shakey, x) ∧ At(b, x) ∧ ¬OnABox

  EFFECT : On(b) ∧ OnABox

)

- Action(

  ClimbDown(b, x),

  PRECOND : Box(b) ∧ At(Shakey, x) ∧ At(b, x) ∧ On(b) ∧ OnABox

  EFFECT : ¬On(b) ∧ ¬OnABox

)

Shakey's Actions (3)

- Action(

  TurnOn(s, x),

  PRECOND : SwitchOff(s) ∧ Switch(s) ∧ At(Shakey, x) ∧ At(s, x) ∧ OnABox

  EFFECT : ¬SwitchOff(s)

)

- Action(

  TurnOff(s, x),

  PRECOND : ¬SwitchOff(s) ∧ Switch(s) ∧ At(Shakey, x) ∧ At(s, x) ∧ OnABox

  EFFECT : SwitchOff(s)

)

Shakey's Initial State

- **Boxes:**
  Box(B1) ∧ Box(B2)

- **Positions:**
  At(Shakey, I) ∧ At(B1, PB1) ∧ At(B2, PB2) ∧ At(S1, PS1) ∧ At(S2, PS2)

- **Switches:**
  Switch(s1) ∧ Switch(s2) ∧ SwitchOn(s2) ∧ SwitchOff(s1)

- **Doors:**
  sameRoom(PB1, PB2) ∧ sameRoom(PB2, PB1) ∧ sameRoom(PB1, PS2) ∧
  sameRoom(PS2, PB1) ∧ sameRoom(PB2, PS2) ∧ sameRoom(PS2, PB2) ∧
  sameRoom(I, PS1) ∧ sameRoom(PS1, I)

- **Goal:** At(B2, I)

Shakey’s Solution

- There is no plan! (No door between the rooms).

PDDL

- PDDL (Planning Domain Definition Language);
- Definition of:
  - Predicates;
  - Actions/Operators;
  - Objects;
  - Initial state;
  - Goal;
- Planning tasks specified in PDDL are separated into:
  - A domain file for predicates and actions;
  - A problem file for objects, initial state and goal specification.

PDDL Files

- Domain file:

```
(define (domain <domain name>)
<PDDL code for predicates>
<PDDL code for first action>
[...]
<PDDL code for last action>
)
```

- Problem file:

```
(define (problem <problem name>)
(:domain <domain name>)
<PDDL code for objects>
<PDDL code for initial state>
<PDDL code for goal specification>
)
```

Shakey Problem in PDDL (1)

- **Predicates:**

```
(:predicates
(at ?x ?y) (on ?x ?y) (in ?x ?y) (box ?x) (turnedOn ?x))
```

- **Actions:**

```
(:action Go
:parameters (?x ?y ?r)
:precondition (and (on Shakey Floor) (at Shakey ?x) (in ?x ?r) (in ?y ?r))
:effect (and (at Shakey ?y) (not (at Shakey ?x))))

(:action Push
:parameters (?b ?x ?y ?r)
:precondition (and (on Shakey Floor) (at Shakey ?x) (box ?b) (at ?b ?x) (in
?x ?r) (in ?y ?r))
:effect (and (at Shakey ?y) (at ?b ?y) (not (at Shakey ?x)) (not (at ?b ?x))))
```

Shakey Problem in PDDL (2)

```
(:action ClimbUp
:parameters (?x ?b)
:precondition (and (on Shakey Floor) (at Shakey ?x) (at ?b ?x) (box ?b))
:effect (and (on Shakey ?b) (not (on Shakey Floor))))

(:action ClimbDown
:parameters (?b)
:precondition (and (on Shakey ?b) (box ?b))
:effect (and (on Shakey Floor) (not (on Shakey ?b))))

(:action TurnOn
:parameters (?s ?b)
:precondition (and (on Shakey ?b) (box ?b) (at ?b ?s))
:effect (turnedOn ?s))
```

Shakey Problem in PDDL (3)

```
(:action TurnOff
:parameters (?s ?b)
:precondition (and (on Shakey ?b) (box ?b) (at ?b ?s))
:effect (not (turnedOn ?s)))
```

- **Objects:**

```
(:objects
Shakey Floor START
Room1 Room2
Door1 Door2
Switch1 Switch2
Box1 Box2
BX1 BX2)
```

Shakey Problem in PDDL (4)

- Initial state:

```
(init:
(on Shakey Floor) (at Shakey START) (in START Room1)
(box Box1) (at Box1 BX1) (in BX1 Room2)
(box Box2) (at Box2 BX2) (in BX2 Room2)
(in Door1 Room1) (in Switch1 Room1) (not (turnedOn Switch1))
(in Door2 Room2) (in Switch2 Room2) (turnedOn Switch2)
```

- Goal:

```
(:goal (and (…) (…))
```

- Get your plan (e.g., use blackbox: https://www.cs.rochester.edu/~kautz/satplan/blackbox/)

PDDL Exercise (in Class)

- Gripper task with four balls:
  - There is a robot that can move between two rooms and pick up or drop balls with either of his two arms. Initially, all balls and the robot are in the first room. We want the balls to be in the second room.
  - Objects: 2 rooms, 4 balls and 2 robot arms.
  - Predicates: room(x), ball(x), at-ball(b, r), free(x), etc.;
  - Initial state: all balls and the robot are in the first room, all robot arms are empty, etc.;
  - Goal specification: all balls must be in the second room.
  - Actions/Operators: the robot can move between rooms, pick up a ball or drop a ball.

STRIPS Exercise (in Class - 1)

- An explorer robot must go on a journey with a jeep. To be able to travel he must: refill the jeep's tank and an fill extra tank to put in the car trunk; put in his backpack a bottle of wine and a sandwich; put the prepared backpack, the binoculars and a map in the jeep. Before filling the bottle he must clean it.

- Suppose that in the initial state the robot is close to the jeep, and all the object and goods required are close to the jeep. The goal state is to have the robot ready to travel.

STRIPS Exercise (in Class – 2)

- Using STRIPS:
  - Describe the initial state.
  - Describe the goal state.
  - Describe the actions.

Partially Ordered Plans (Recap)

- How to <u>practically</u> get POPs:
  1. Initial plan, containing two steps called «start» and «finish»:
     - «start» has the initial conditions as its effects;
     - «finish» has the goal conditions as its preconditions;
  2. Iterate until the plan is *complete* (i.e., every precondition at each step is satisfied by the effect of some other step):
     a. <u>Choose</u> a *subgoal* (i.e., a precondition *p* not yet satisfied);
     b. <u>Choose</u> an operator that can satisfy *p*;
     c. Check that the work done in the previous step doesn't violate any of our previous causal links: if it happens, resolve the threat.

POPs: Choose (Recap)

- We want *p* to be true:
  - Find some step already in the existing plan, having *p* as an effect OR add a new step to the plan;
- Choose: arbitrary choose for execution one operator within a set of possible things that could be done;
- Fail: if you fail (i.e., there is no operator that makes *p* true) just go back to the most recent choice and try a different option;
  - If they have all been tried: go recursively (i.e., go back to the most recent choice);
- If you fail up to the top level of the program, there is no plan.

POPs: Causal Links (Recap)

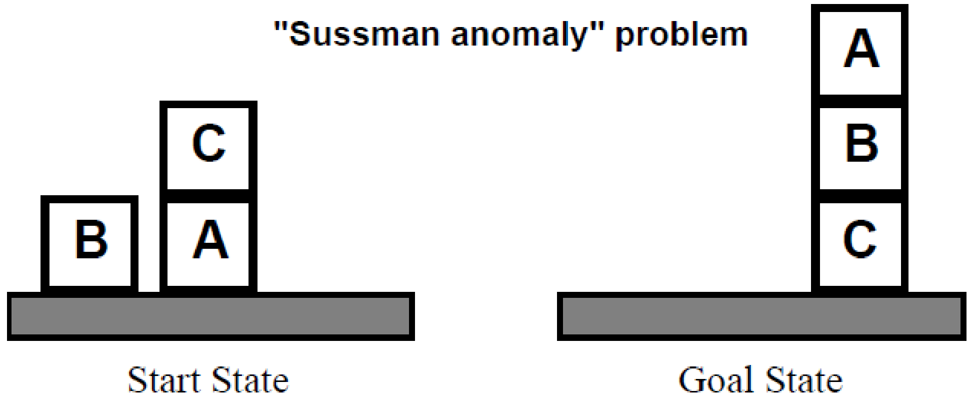
- If you find an operator that can make *p* true:
  - Choose that step from the already available plan or add a new step to the plan by instantiating that operator;
  - Add a causal link between the two steps;
  - Add an ordering constraint between the two steps (the new step goes before the one requiring *p*);
  - If needed:
    - Add a variable binding constraint (e.g., for binding a variable in the new step with *p*);
    - If the new constraint is incompatible with the existing constraints, fail (i.e., go back to…).

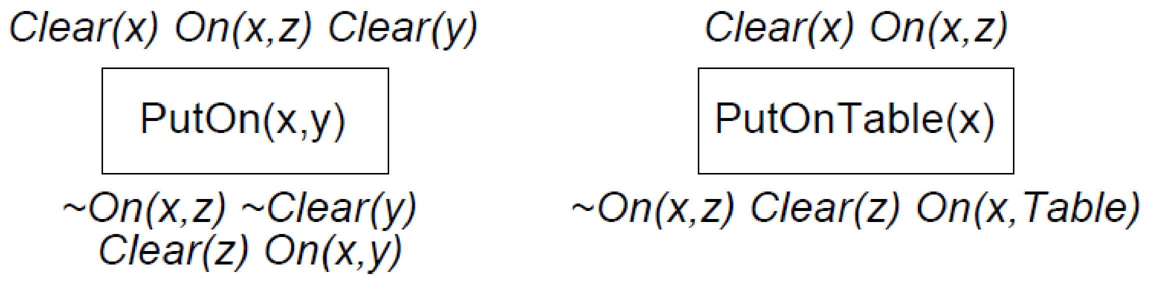
POPs: Resolve Threats (Recap)

- A step *s* could violate a causal link by undoing the «work» previously done to estabilish a certain value of *p*, making it impossible to execute successive operators;
- How to deal with threats:
  - Promotion: move *s* in such a way that it comes before the operator establishing the value of *p*;
  - Demotion: move *s* in such a way that it comes after the value of *p* is used for the execution of successive operators;
  - If promotion fails, try demotion;
  - If both fail, try a different way to satisfy *p*;
  - If variables appear in the operator, try to constraint the variables in such a way that it has not negative effects in *s*;

POP Example (1)

- Define the ordering constraints and the causal links among the actions, and produce a partially ordered plan to solve the problem.

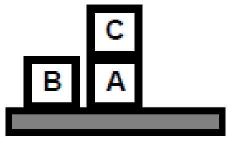




POP Example (2)

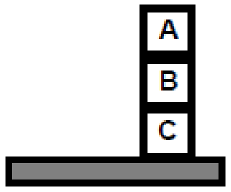
START

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

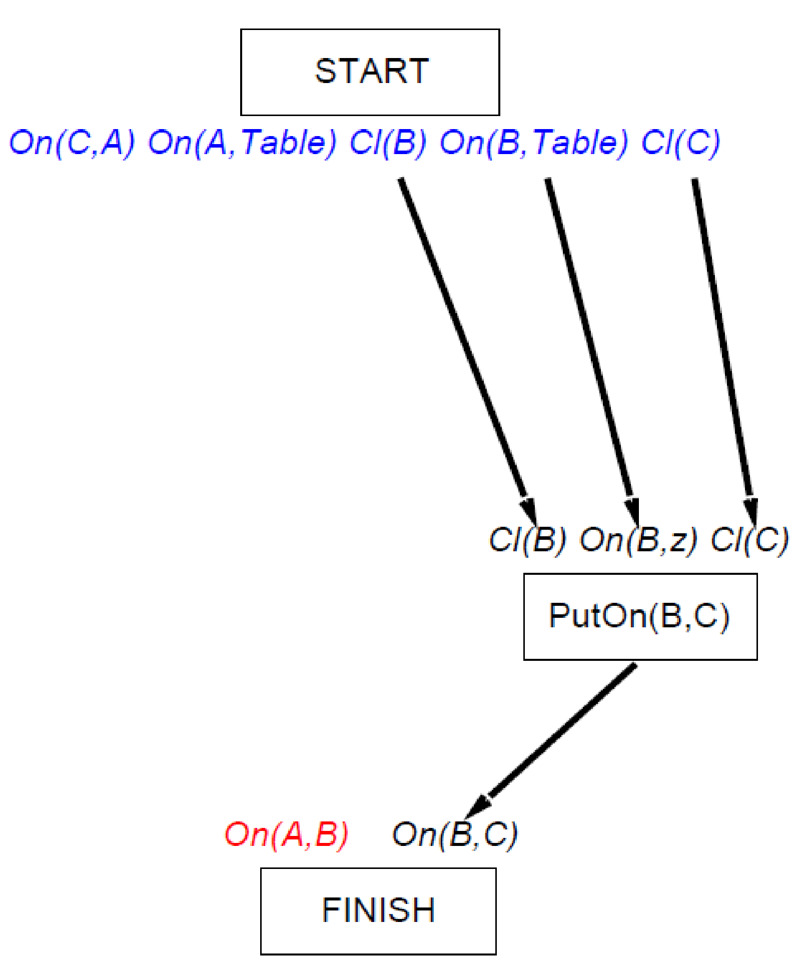


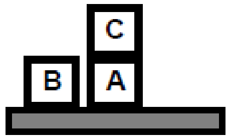
On(A,B) On(B,C)

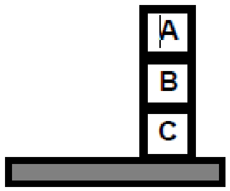
FINISH



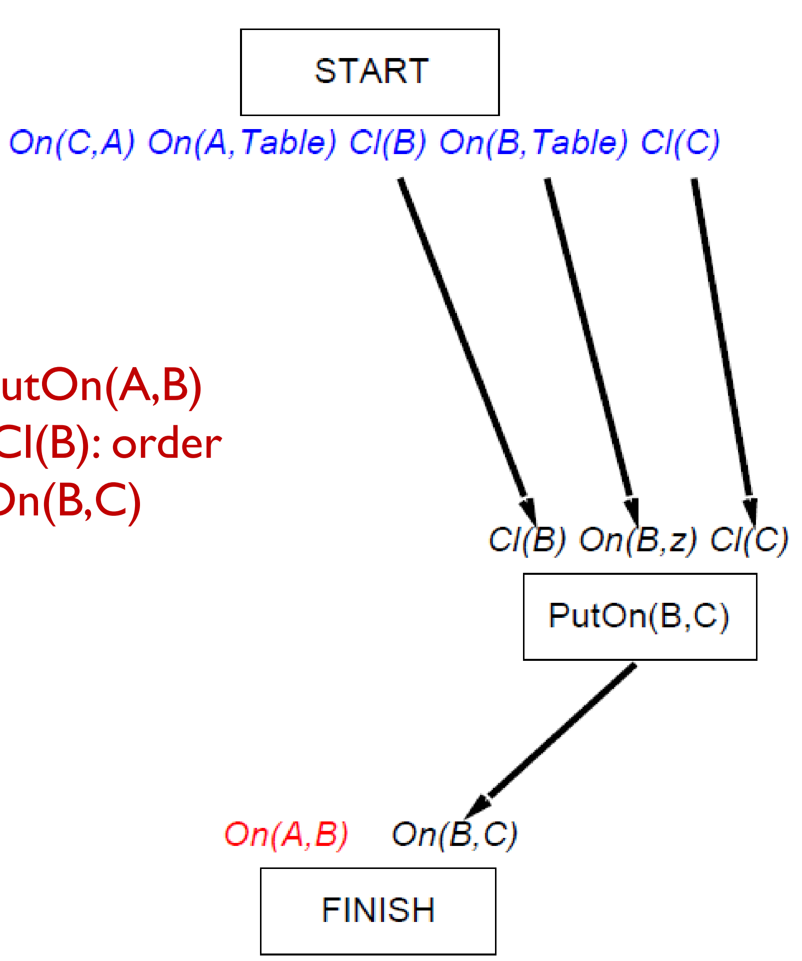
POP Example (3)

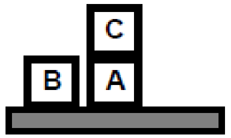




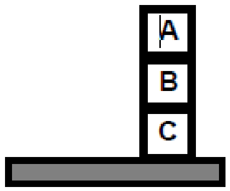


POP Example (3)

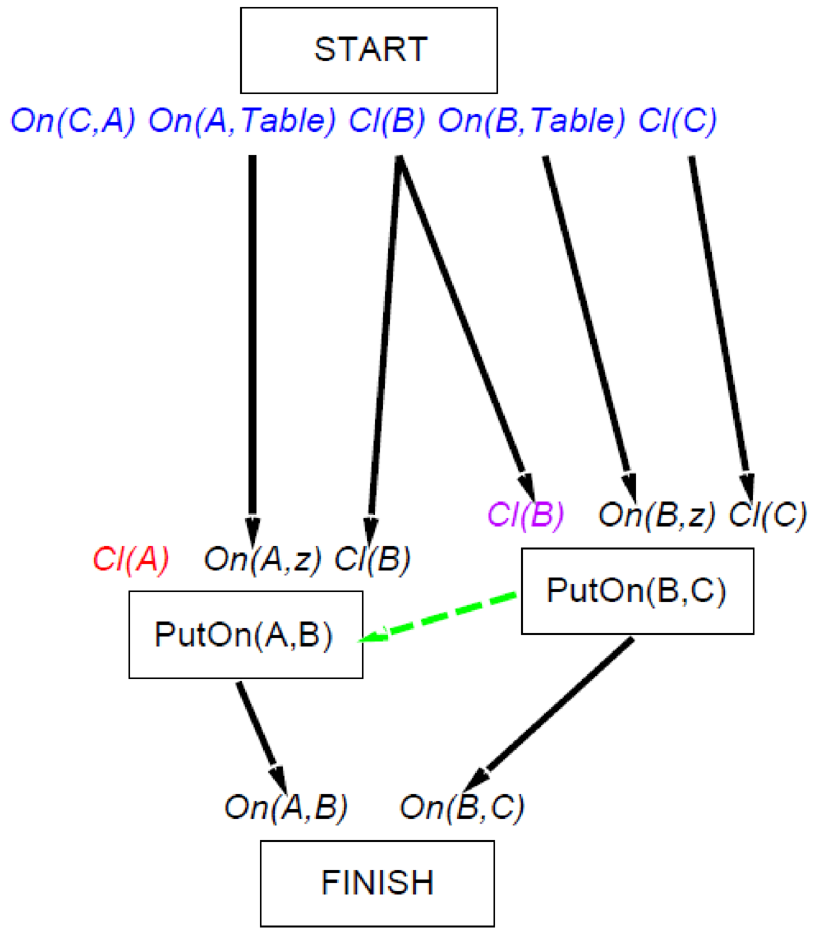


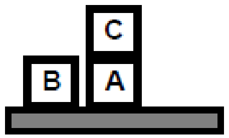


NOTE: PutOn(A,B) clobbers Cl(B): order after PutOn(B,C)

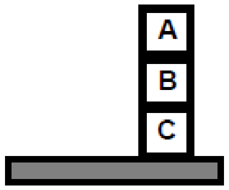


POP Example (4)

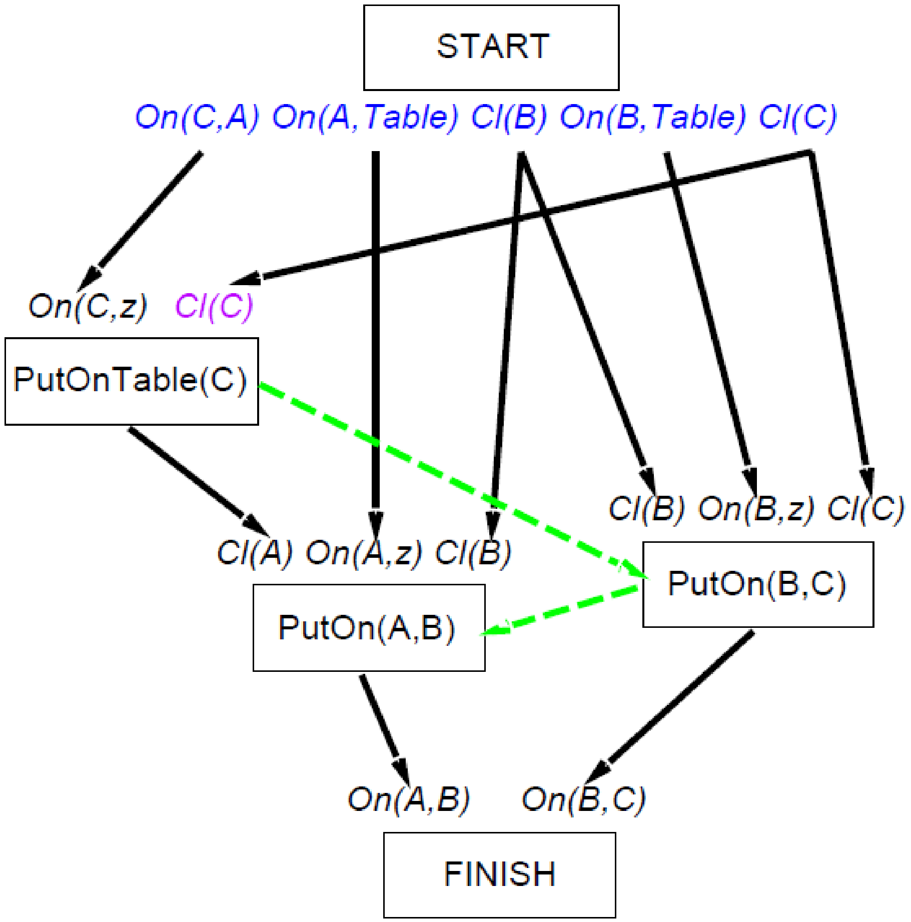


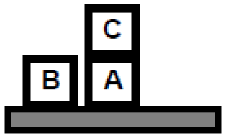


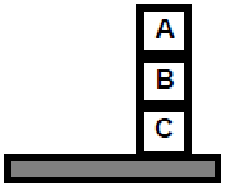
NOTE: PutOn(B,C) clobbers Cl(C): order after PutOnTable(C)



POP Example (5)







POP Exercise (in Class)

- Produce the POP for the following problem:

- Action

  Buy(x, store)

  PRECOND: At(store), Sells(store, x)

  EFFECT: Have(x)

  Go(x, y)

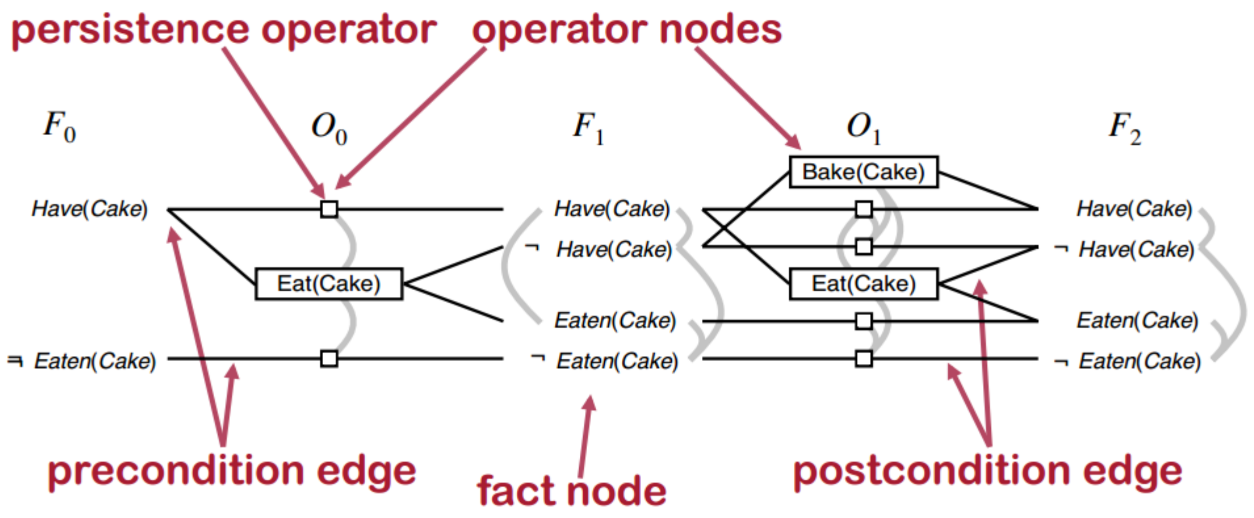
  PRECOND: At(x)

  EFFECT: At(y), ¬At(x)

- Goal

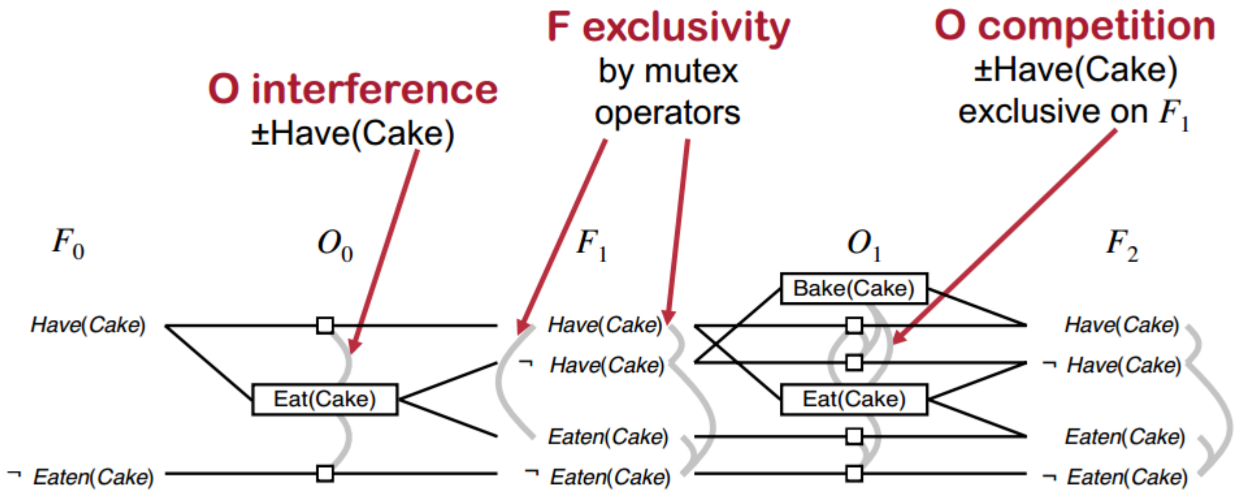
  Have(Milk) ∧ Have(Banana) ∧ Have(Drill)

Planning Graphs (Recap - 1)



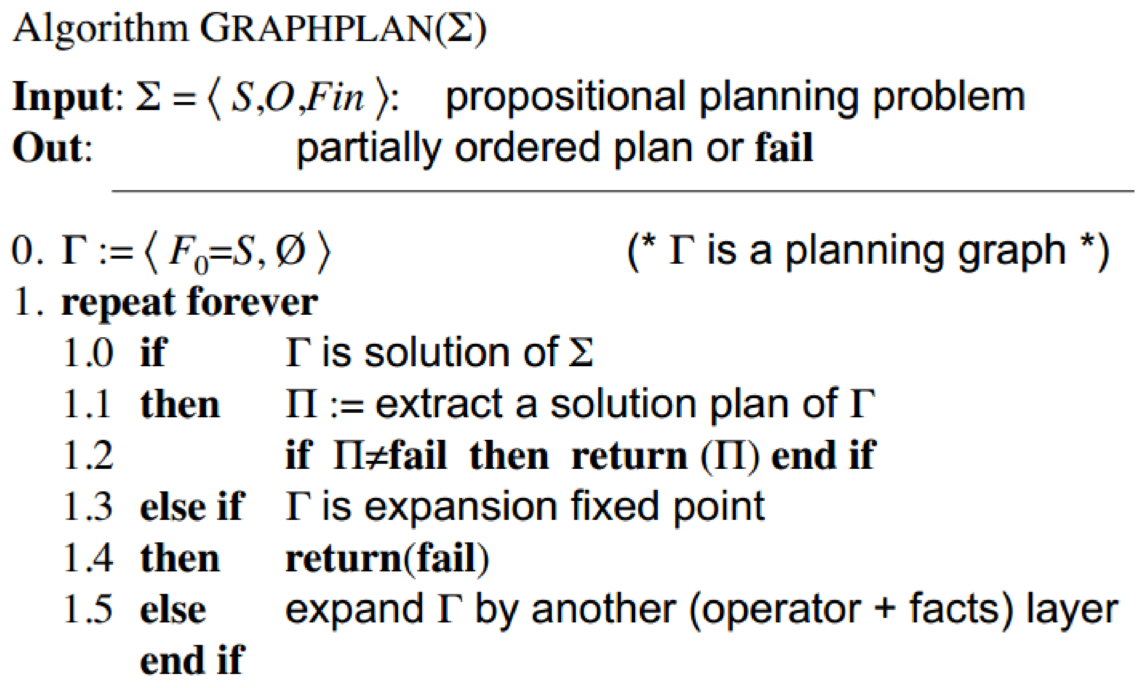
- Two (different!) operators in a layer are exclusive:
  - in layer O_0 iff they interfere;
  - in layer O_i>0 iff they interfere or compete;
- Two operators interfere iff one destroys a precondition or an effect of the other;
- Two operators compete in layer O_i, iff they have facts among their preconditions that are exclusive on the previous layer F_i;
- Two facts f, g are exclusive in layer F_i>0 iff there are only exclusive operators in the previous layer O_{i-1} that produce f and g.

Planning Graphs (Recap – 2)



- Generate "solving" planning graph;
- Extract mutex-free plan;
- If no mutex-free plan found, extend planning graph;
- Expansion of a planning graph G_n= 〈N,E〉 of the order n: add operator layer O_n, fact layer F_{n+1}, and mutex arcs;
- An expansion fixed point is a planning graph, in which two fact layers and all their mutex conditions are identical.

Planning Graphs (Recap – 3)



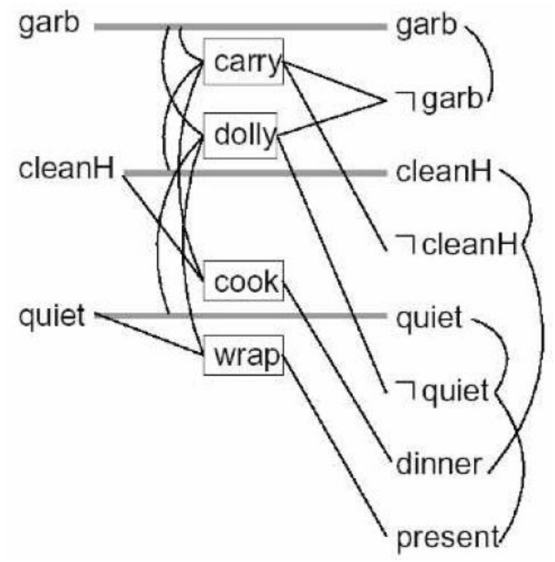
```
Algorithm GRAPHPLAN($\Sigma$)

Input: $\Sigma = \langle S, O, Fin \rangle$:   propositional planning problem
Out:                    partially ordered plan or fail
________________________________________________________
0. $\Gamma := \langle F_0 = S, \emptyset \rangle$                 (* $\Gamma$ is a planning graph *)
1. repeat forever
    1.0  if        $\Gamma$ is solution of $\Sigma$
    1.1  then      $\Pi$ := extract a solution plan of $\Gamma$
    1.2            if $\Pi \neq$ fail then return ($\Pi$) end if
    1.3  else if   $\Gamma$ is expansion fixed point
    1.4  then      return(fail)
    1.5  else      expand $\Gamma$ by another (operator + facts) layer
         end if
```

Planning Graphs Example (1)

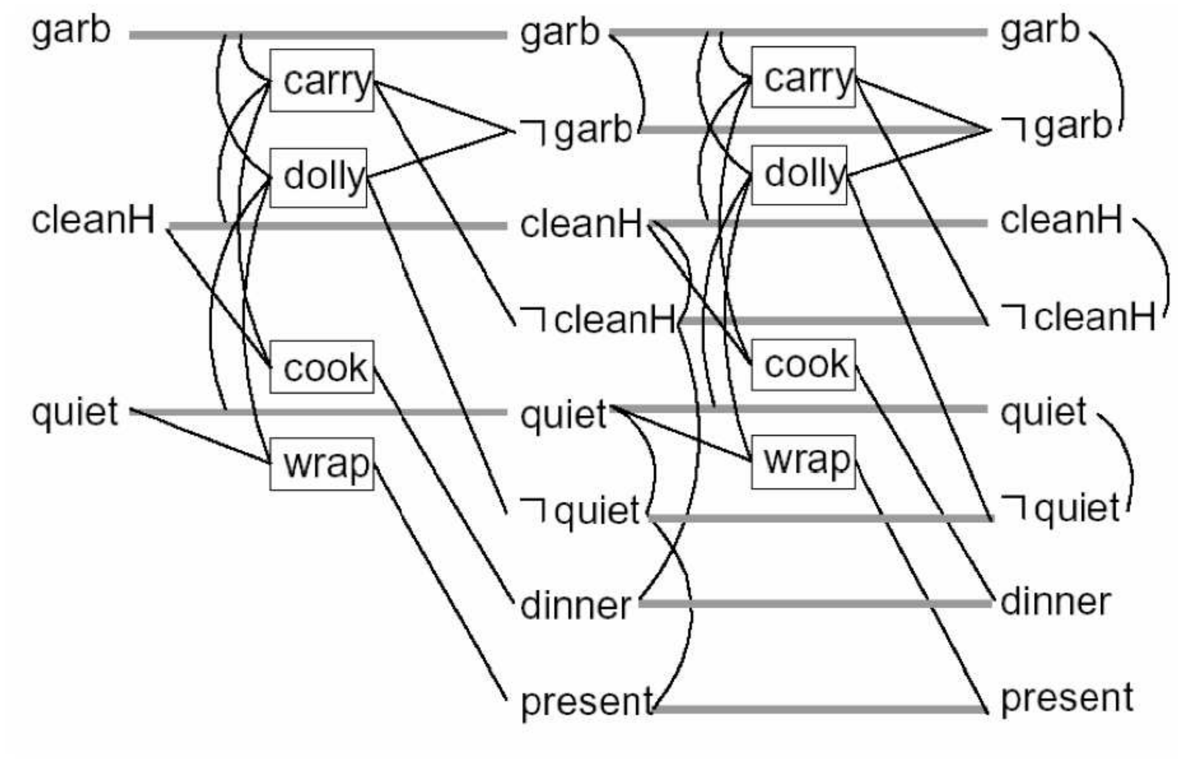
- **Initial state:** garbage ∧ cleanHands ∧ quiet
- **Goal state:** dinner ∧ present ∧ ¬garbage
- **Actions:**
  - Cook:
    - PRECOND: cleanHands;
    - EFFECT: dinner
  - Wrap:
    - PRECOND: quiet;
    - EFFECT: present
  - Carry:
    - EFFECT: ¬garbage ∧ ¬cleanHands
  - Dolly:
    - EFFECT: ¬garbage ∧ ¬quiet

Planning Graphs Example (2)



- wrap and dolly are mutex because dolly negates the precondition of wrap;
- carry and the no-op are mutex because one negates the effect of the other;
- present and ¬quiet are mutex because the actions achieving them, wrap and dolly, are mutex.

Planning Graphs Example (3)



Thank you!