

*16-350*

*Planning Techniques for Robotics*

*Search Algorithms:  
Planning on Symbolic Representations*

*Maxim Likhachev*

*Robotics Institute*

*Carnegie Mellon University*

# We are given a problem; need to compute a plan

- STRIPS representation of the problem



## ***Start state:***

$On(A,B) \wedge On(B,Table) \wedge On(C,Table) \wedge Block(A) \wedge Block(B) \wedge Block(C) \wedge Clear(A) \wedge Clear(C)$

## ***Goal state:***

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

## ***Actions:***

$MoveToTable(b,x)$

*Precond:*  $On(b,x) \wedge Clear(b) \wedge Block(b)$

*Effect:*  $On(b,Table) \wedge Clear(x) \wedge \sim On(b,x)$

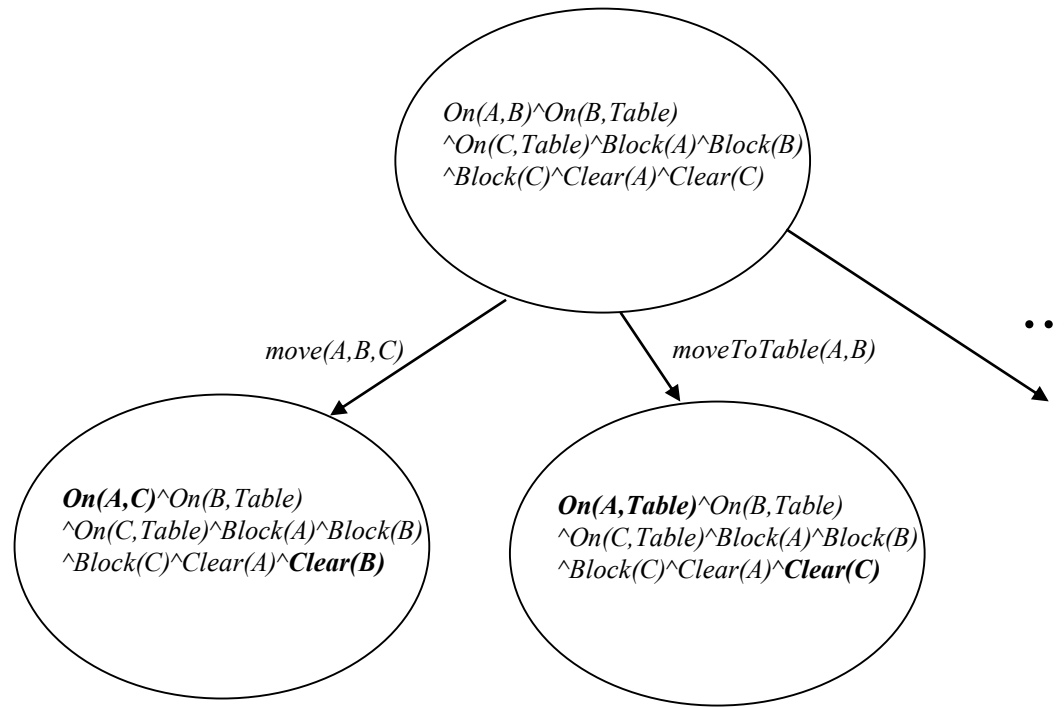
$Move(b,x,y)$

*Precond:*  $On(b,x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b) \wedge Block(y) \wedge (b \neq y)$

*Effect:*  $On(b,y) \wedge Clear(x) \wedge \sim On(b,x) \wedge \sim Clear(y)$

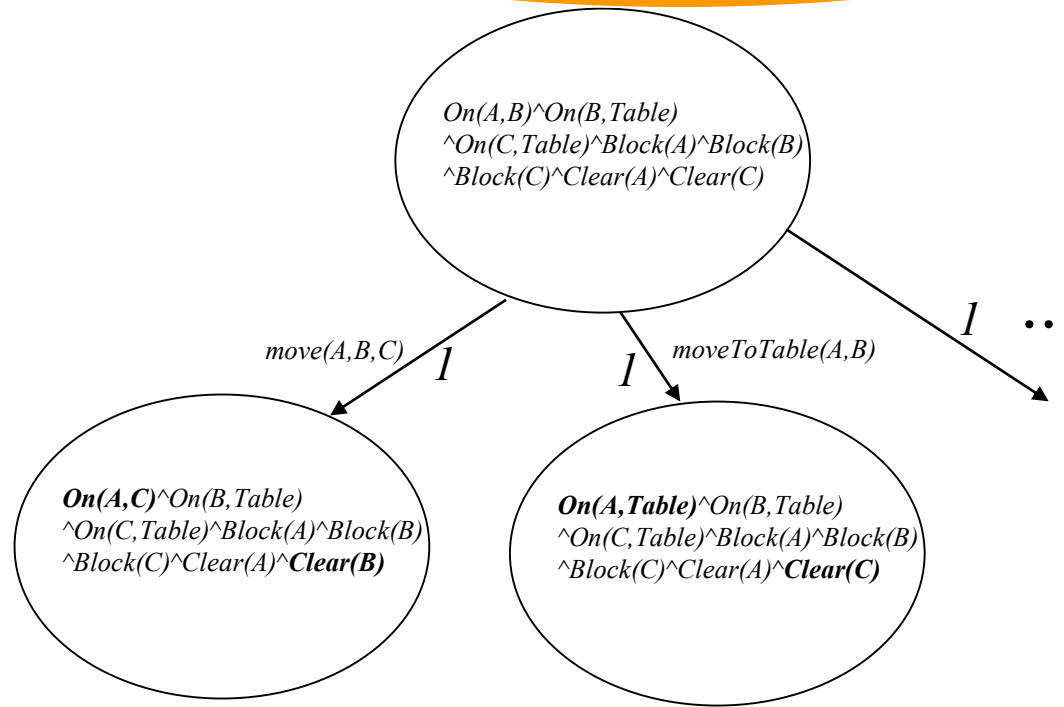
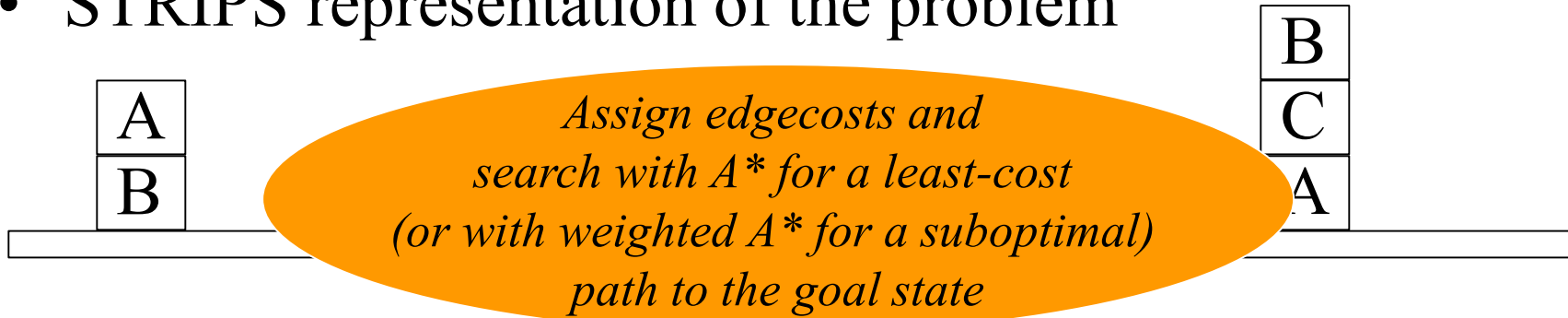
# Planning via Graph Search

- STRIPS representation of the problem



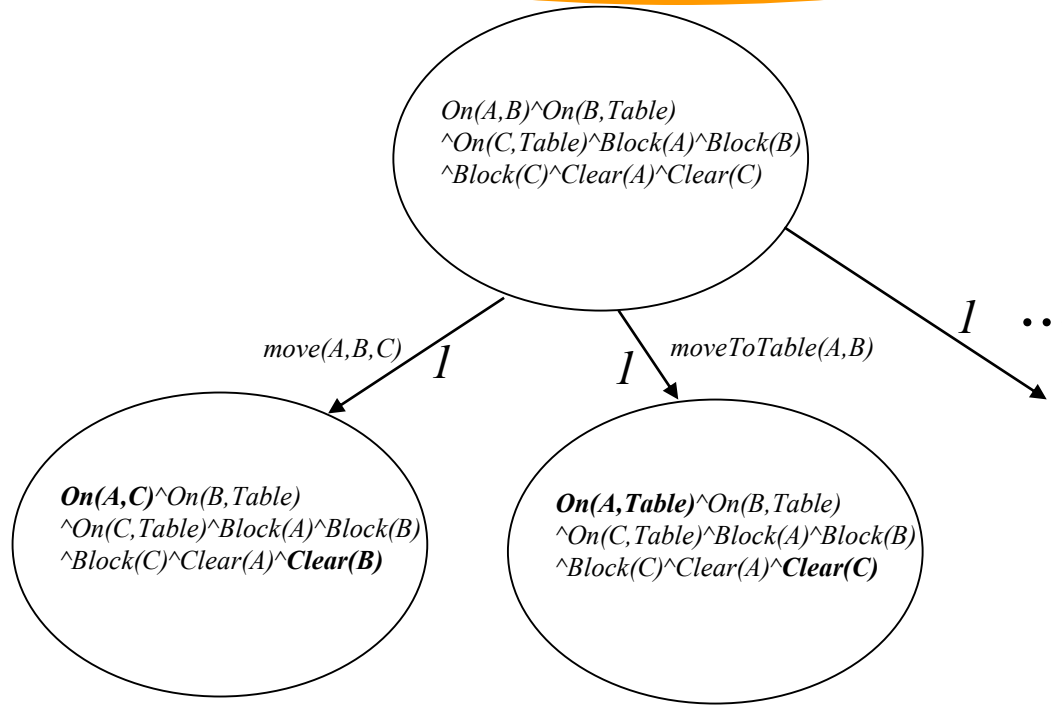
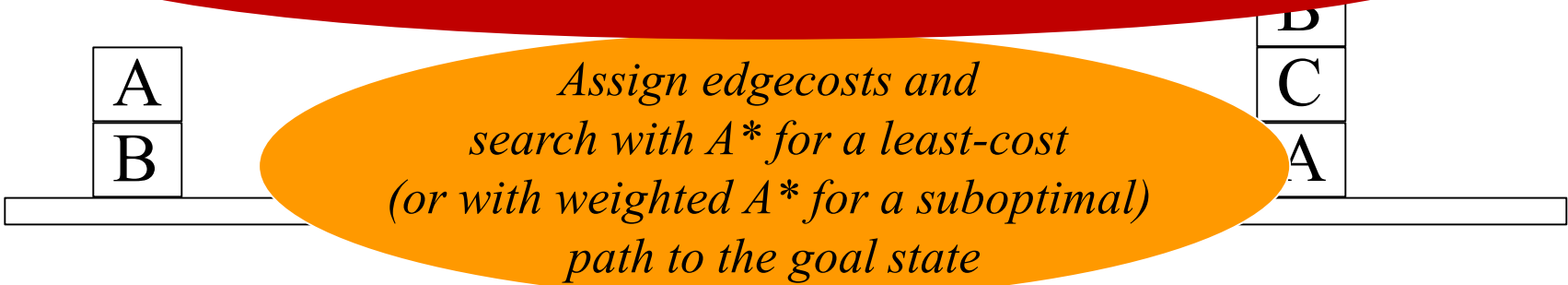
# Planning via Graph Search

- STRIPS representation of the problem



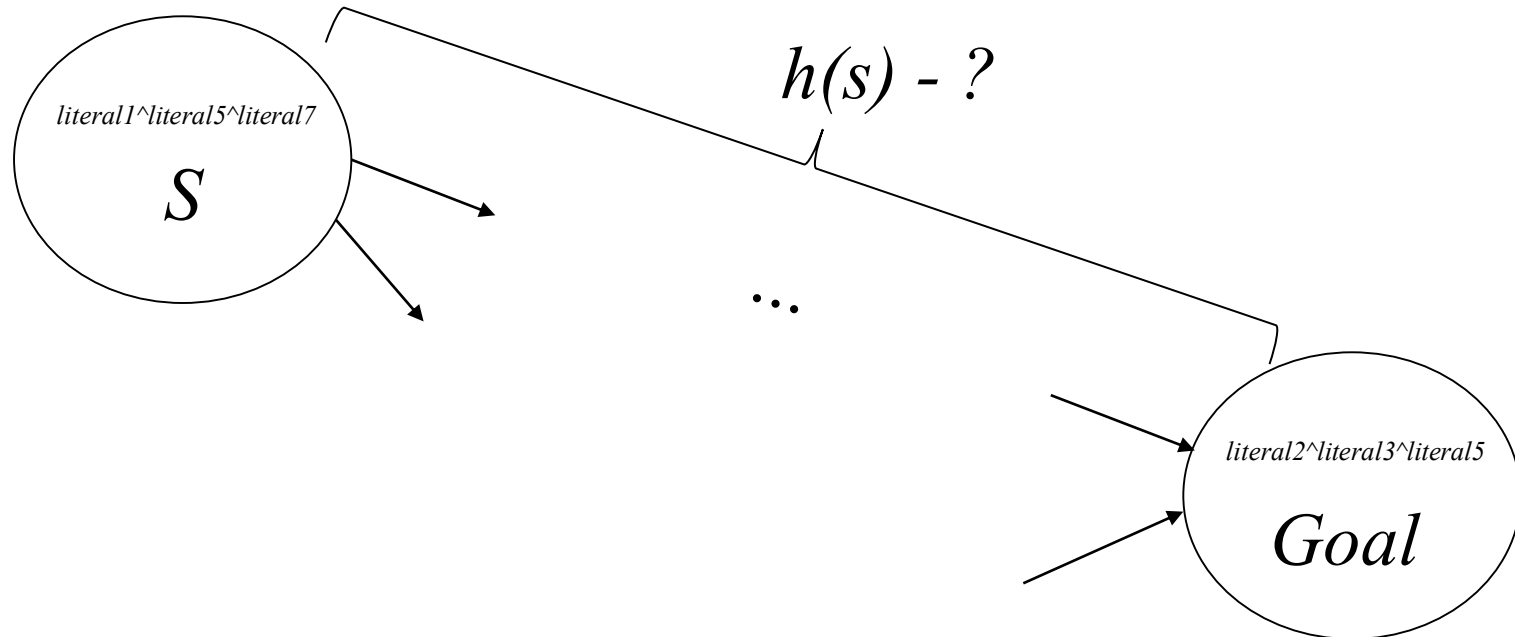
# Planning via Graph Search

- How do we compute *domain-independent heuristics*?



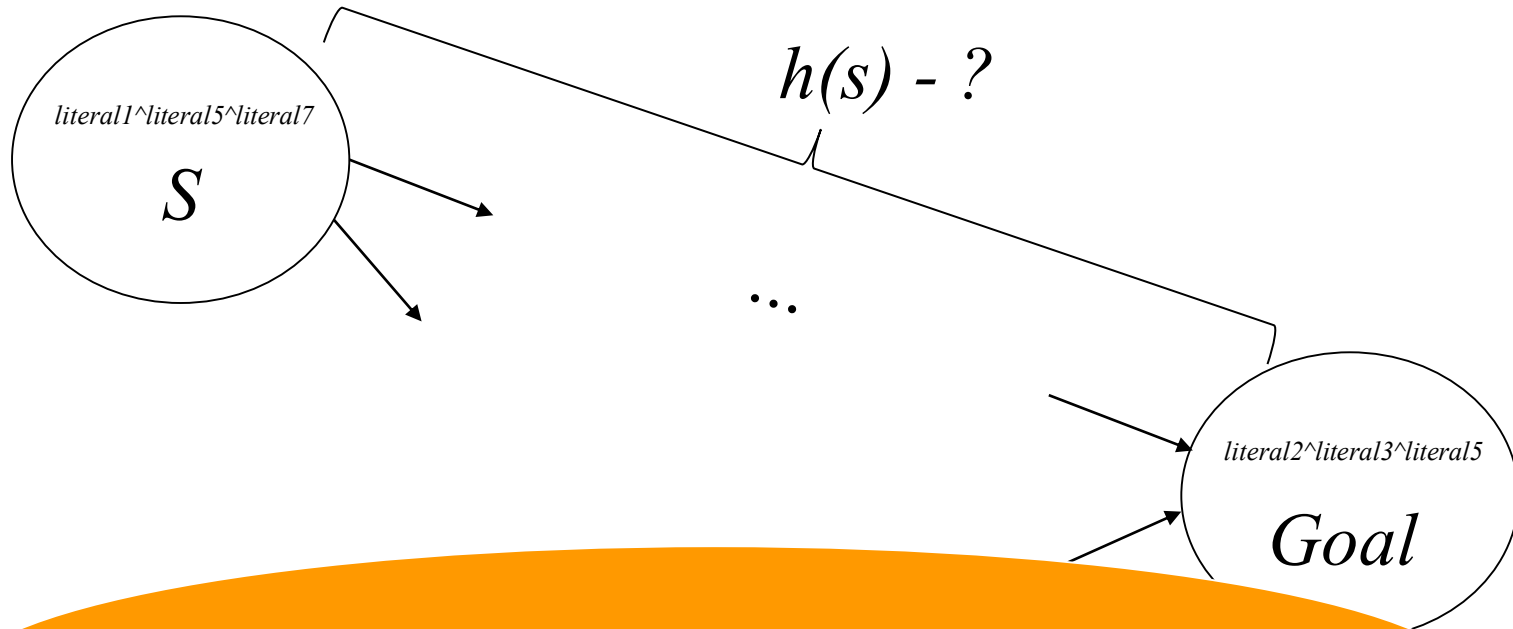
# Planning via Graph Search

- Computing heuristics



# Planning via Graph Search

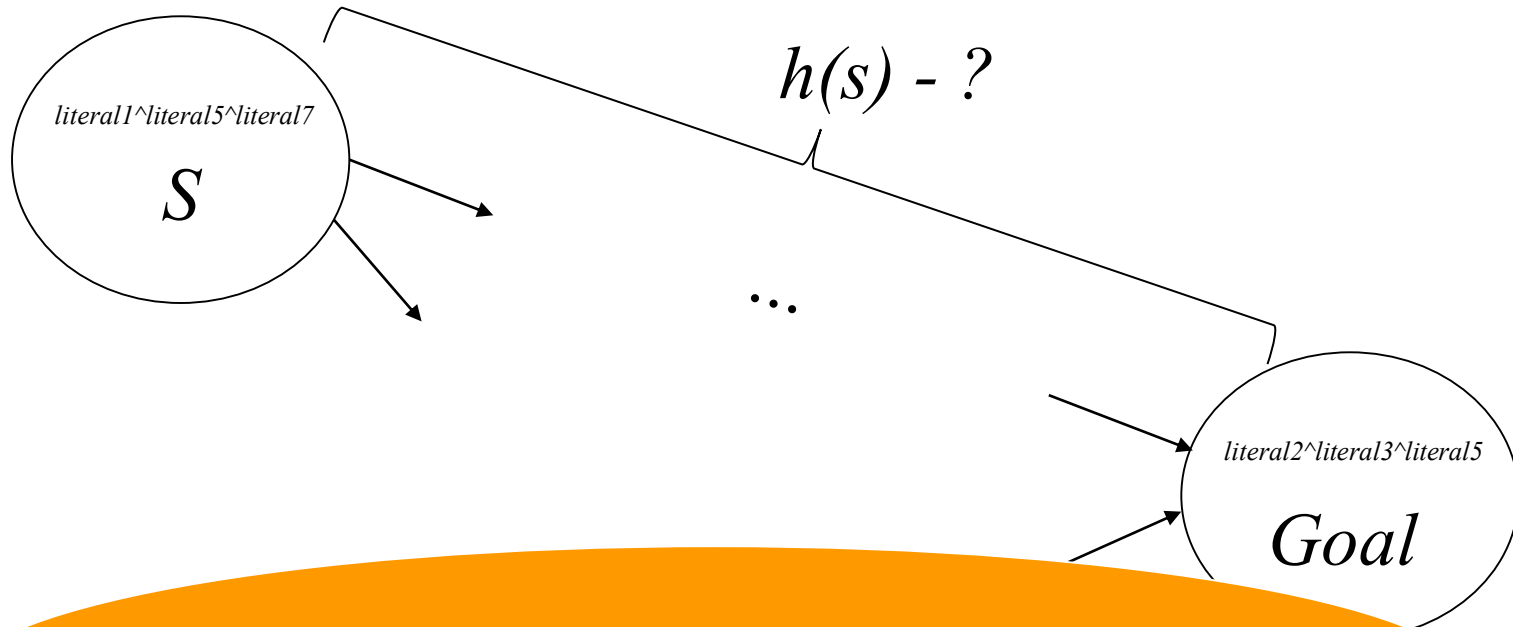
- Computing heuristics



*Option 1:  $h(s) = \#$  of literals that are NOT yet satisfied  
i.e.,  $h(s) = \#$  of literals  $l_i$  such that  $l_i(s) = false$  and  $l_i(goal) = true$*

# Planning via Graph Search

- Computing heuristics



*Option 1:  $h(s) = \#$  of literals that are NOT yet satisfied  
i.e.,  $h(s) = \#$  of literals  $l_i$  such that  $l_i(s) = \text{false}$  and  $l_i(\text{goal}) = \text{true}$*

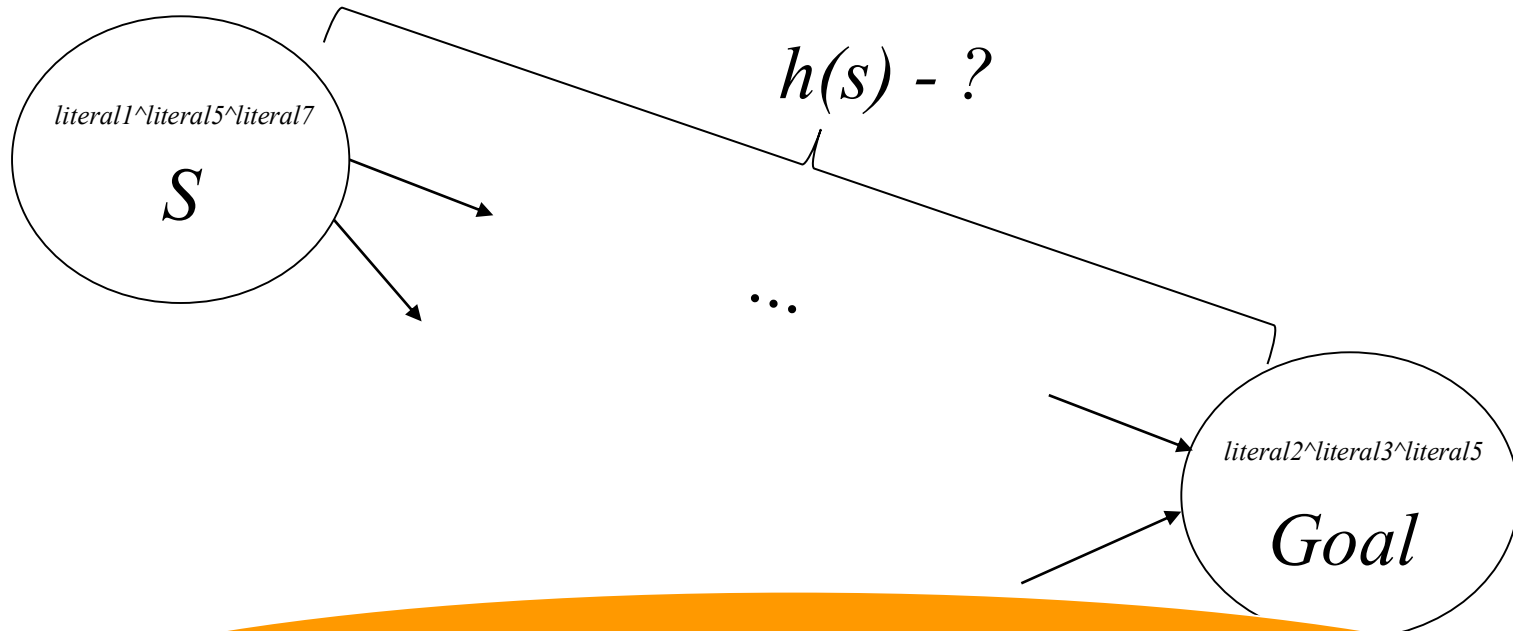
*Is this heuristic function admissible?*

*Can we still use it? What do we sacrifice?*



# Planning via Graph Search

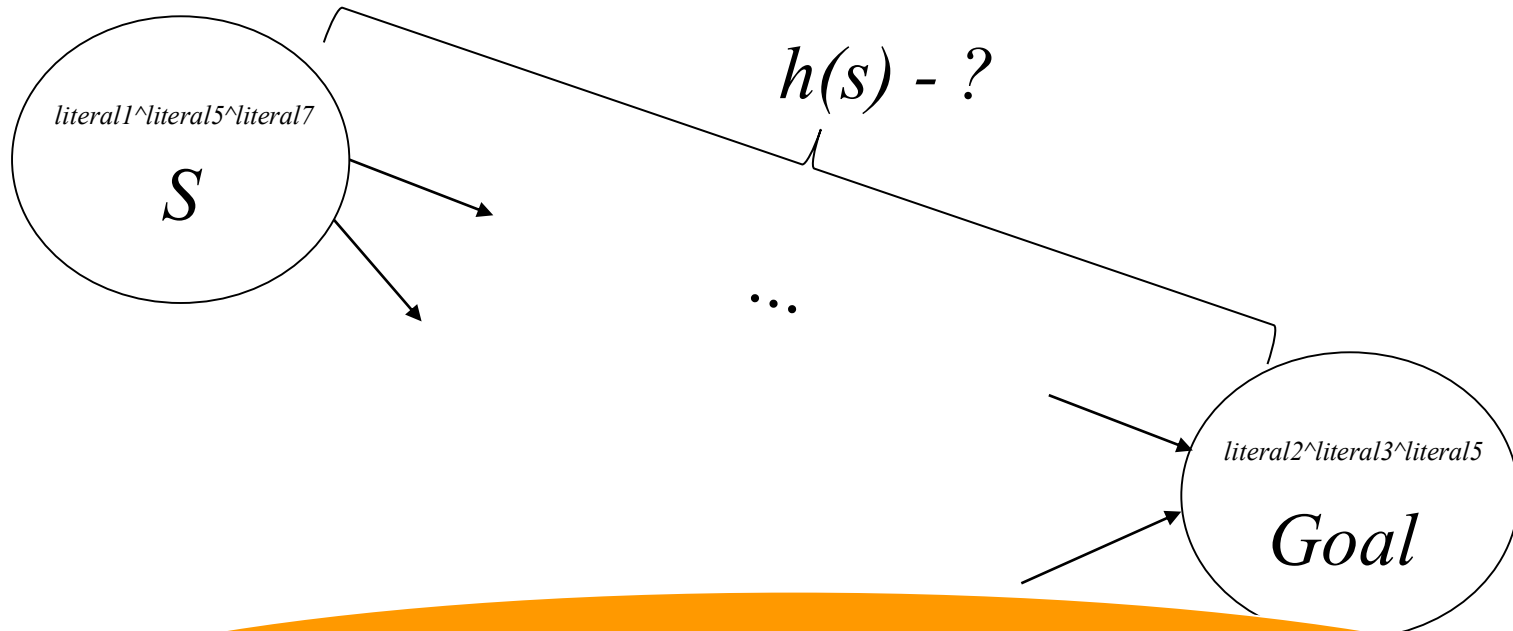
- Computing heuristics



*Option 2: compute heuristics using a **relaxed** (simpler) problem  
Common relaxation: assume actions don't have any negative effects  
(called empty-delete-list heuristics)*

# Planning via Graph Search

- Computing heuristics



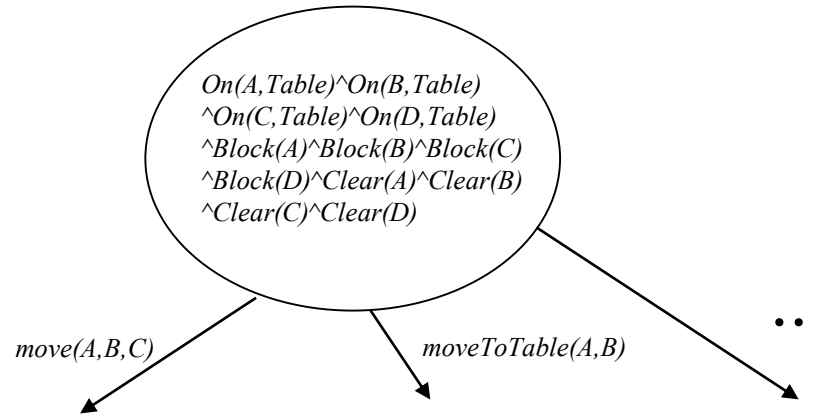
*Option 2: compute heuristics using a **relaxed** (simpler) problem  
Common relaxation: assume actions don't have any negative effects  
(called empty-delete-list heuristics)*

*Any downsides?*

*Despite computational complexity,  
still very popular as it speeds the overall search tremendously*

# Planning via Graph Search

- Challenges in graph search formulation



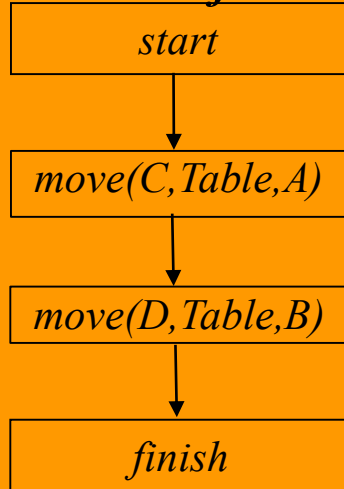
*How many successors?*

# Planning via Graph Search

- Challenges in graph search formulation



*The plan we find is a total order of actions:*



$On(A, Table) \wedge On(B, Table)$   
 $\wedge On(C, Table) \wedge On(D, Table)$   
 $\wedge Block(A) \wedge Block(B) \wedge Block(C)$   
 $\wedge Block(D) \wedge Clear(A) \wedge Clear(B)$   
 $\wedge Clear(C) \wedge Clear(D)$

*moveToTable(A, B)*

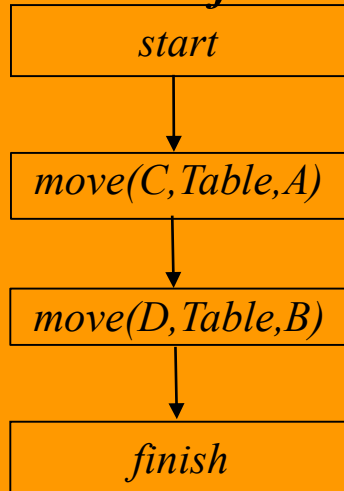
...

# Planning via Graph Search

- Challenges in graph search formulation



*The plan we find is a total order of actions:*



$On(A, Table) \wedge On(B, Table)$   
 $\wedge On(C, Table) \wedge On(D, Table)$   
 $\wedge Block(A) \wedge Block(B) \wedge Block(C)$   
 $\wedge Block(D) \wedge Clear(A) \wedge Clear(B)$   
 $\wedge Clear(C) \wedge Clear(D)$

*moveToTable(A, B)*

...

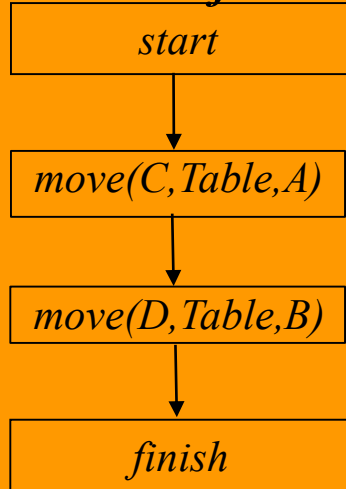
*Does it have to be a total order?*

# Partial-Order Planning (POP)

- Total vs. partial ordering of actions



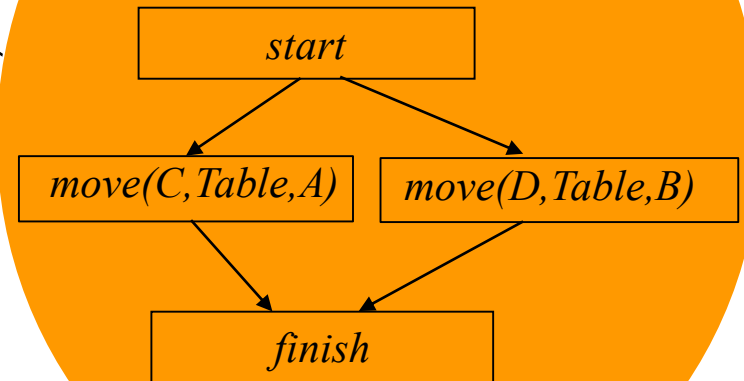
*The plan we find is a total order of actions:*



$On(A, Table) \wedge On(B, Table)$   
 $\wedge On(C, Table) \wedge On(D, Table)$   
 $\wedge Block(A) \wedge Block(B) \wedge Block(C)$   
 $\wedge Block(D) \wedge Clear(A) \wedge Clear(B)$   
 $\wedge Clear(C) \wedge Clear(D)$

*moveToTable(A,B)*

*POP aims to compute a partial order of actions:*

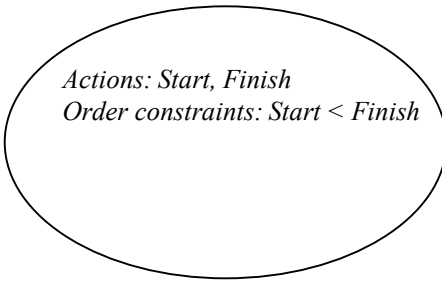


# Partial-Order Planning (POP)

- Searches the space of “plans”
  - State defined by:
    - The currently selected set of actions
    - Set of ordering constraints in the form of  $A < B$  (action A has to be executed at some point before action B). No cycles allowed (i.e.,  $A < B$  and  $B < A$  is a cycle and makes such state invalid)
    - Set of causal links in the form of  $A \stackrel{p}{=} > B$  (action A achieves precondition  $p$  required by action B)

# Partial-Order Planning (POP)

- Searches the space of “plans”
  - State defined by:
    - The currently selected set of actions
    - Set of ordering constraints in the form of  $A < B$  (action A has to be executed at some point before action B). No cycles allowed (i.e.,  $A < B$  and  $B < A$  is a cycle and makes such state invalid)
    - Set of causal links in the form of  $A \stackrel{p}{=} > B$  (action A achieves precondition  $p$  required by action B)



*Actions: Start, Finish*  
*Order constraints: Start < Finish*

*Start state*



# Partial-Order Planning (POP)

- Searches the space of “plans”
  - State defined by:
    - The currently selected set of actions
    - Set of ordering constraints in the form of  $A < B$  (action A has to be executed at some point before action B). No cycles allowed (i.e.,  $A < B$  and  $B < A$  is a cycle and makes such state invalid)
    - Set of preconditions  $p$

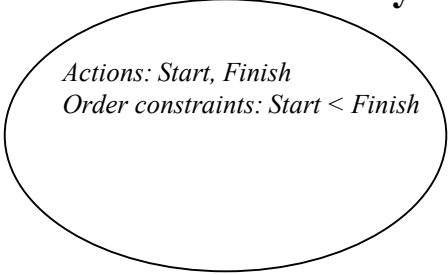
*Start action has: no preconditions; effect=the literals in the actual start state*  
*Finish action has: preconditions=the literals in the actual goal state; no effects*

*Actions: Start, Finish*  
*Order constraints: Start < Finish*

*Start state*

# Partial-Order Planning (POP)

- Searches the space of “plans”
  - Successor  $S'$  of state  $S$  computed as follows:
    - Pick any action  $B$  in  $S$  which has at least one precondition  $p$  not satisfied
    - Choose any action  $A$  (either a new action or an existing action in state  $S$ ) that achieves  $p$  and
      - Add  $A$  to  $S'$  (if not in it already)
      - Add  $A < B$ ,  $Start < A$ ,  $A < Finish$  orders to  $S'$
      - Add  $A \stackrel{p}{=} B$  causal link to  $S'$
      - If any other action  $C$  in  $S'$  removes  $p$ , then  $C < A$  or  $B < C$  constraint added
      - If  $A$  removes precondition  $p'$  used in a causal link  $D \stackrel{p'}{=} F$ , then  $A < D$  or  $F < A$  added
      - **If any constraint cycle is introduced, then  $S'$  is an invalid successor**



Actions:  $Start$ ,  $Finish$   
Order constraints:  $Start < Finish$

*Start state*

# Partial-Order Planning (POP)

- Searches the space of “plans”
  - Successor  $S'$  of state  $S$  computed as follows:
    - Pick any action  $B$  in  $S$  which has at least one precondition  $p$  not satisfied
    - Choose any action  $A$  (either a new action or an existing action in state  $S$ ) that achieves  $p$  and
      - Add  $A$  to  $S'$  (if not in it already)
      - Add  $A < B$ ,  $Start < A$ ,  $A < Finish$  orders to  $S'$
      - Add  $A \stackrel{p}{=} > B$  causal link to  $S'$
      - If any other action  $C$  in  $S'$  removes  $p$ , then  $C < A$  or  $B < C$  constraint added
      - If  $A$  removes precondition  $p'$  used in a causal link  $D \stackrel{p'}{=} > F$ , then  $A < D$  or  $F < A$  added
      - **If any constraint cycle is introduced, then  $S'$  is an invalid successor**

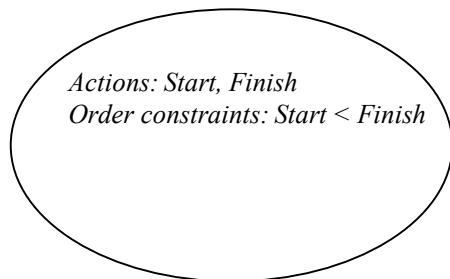
Actions: Start, Finish  
Order constraints: Start < Finish

Start state

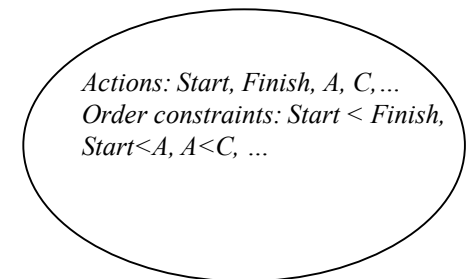
*This gives us an implicit graph  
that is typically searched by Depth-First Search  
for any feasible solution to the goal state*

# Partial-Order Planning (POP)

- Searches the space of “plans”
  - Terminate the search as soon as a state where all actions have all their preconditions met is reached (e.g., a goal state of the search)



*Start state*



*Goal state*

# Partial-Order Planning (POP)

- Searches the space of “plans”
  - Terminate the search as soon as a state where all actions have all their preconditions met is reached (e.g., a goal state of the search)



*Actions: Start, Finish*  
*Order constraints: Start < Finish*

*Start state*

*Example on the board*

# What You Should Know...

---

- How to compute domain-independent heuristics
- Motivation for Partial-order Planning
- The general idea behind how Partial-order Planning works