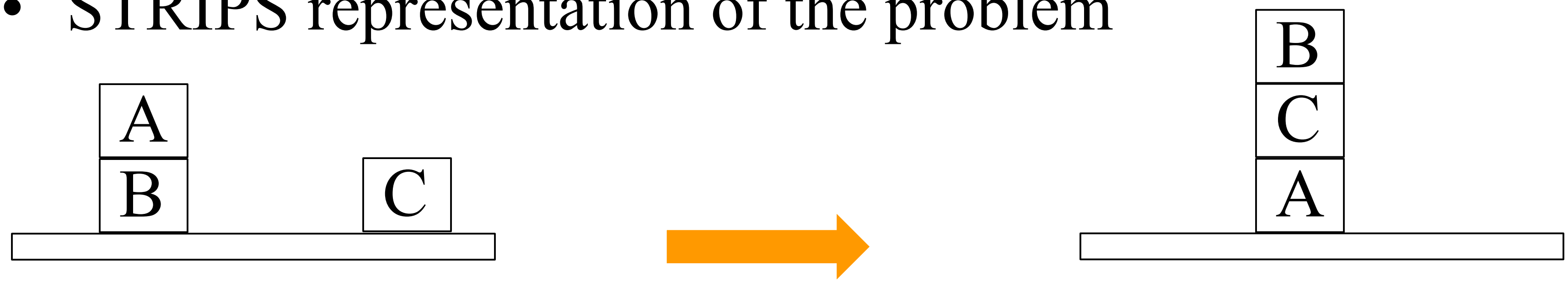
16-782
Planning & Decision-making in Robotics

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)^On(B,Table)^On(C,Table)^Block(A)^Block(B)^Block(C)^Clear(A)^Clear(C)

Goal state:

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

Actions:

MoveToTable(b,x)

Precond: On(b,x)^Clear(b)^Block(b)^Block(x)

Effect: On(b,Table)^Clear(x)^~On(b,x)

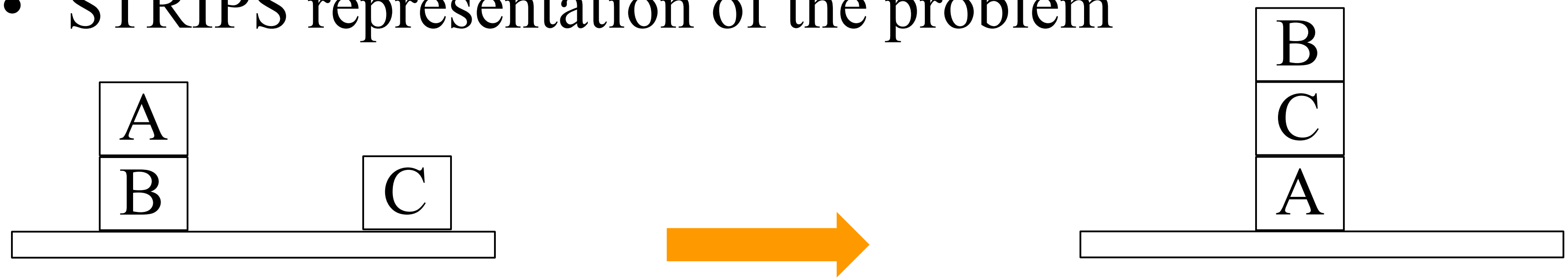
Move(b,x,y)

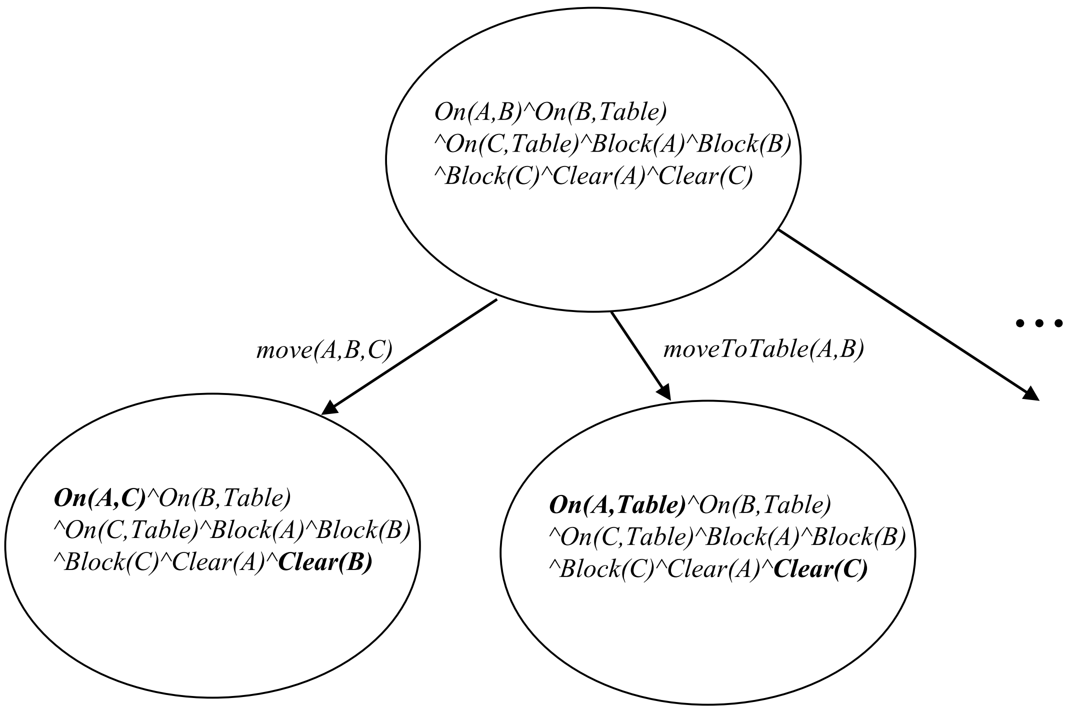
Precond: On(b,x)^Clear(b)^Clear(y)^Block(b)^Block(y)^(b~=y)

Effect: On(b,y)^Clear(x)^~On(b,x)^~Clear(y)

Planning via Graph Search

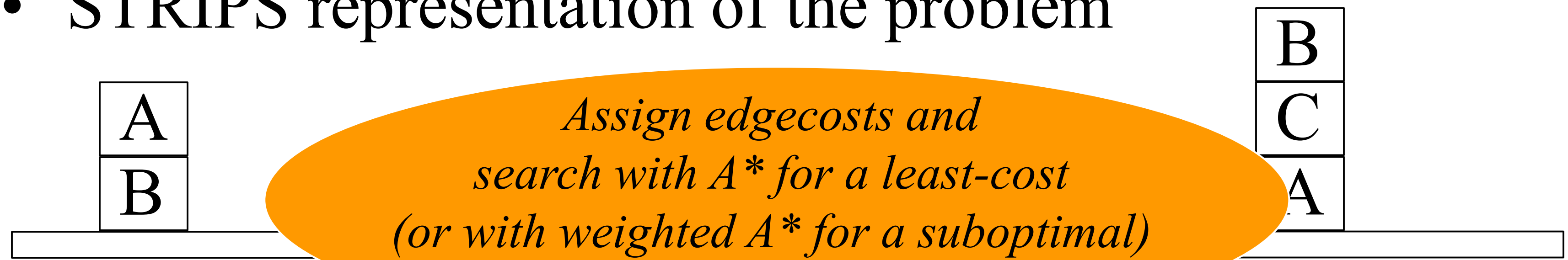
- STRIPS representation of the problem



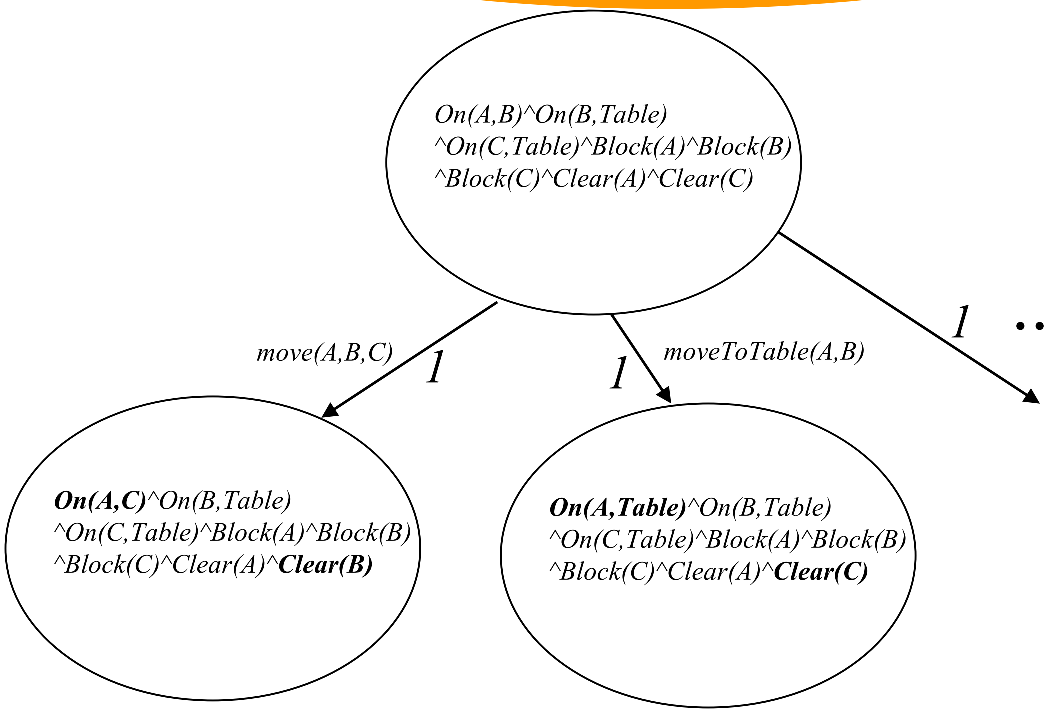


Planning via Graph Search

- STRIPS representation of the problem

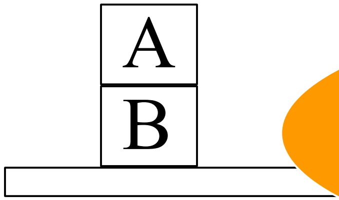


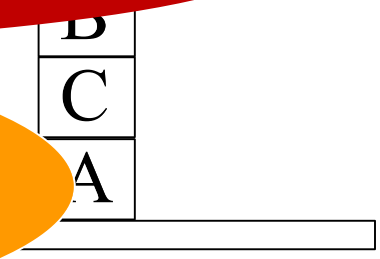
Assign edgecosts and search with A for a least-cost (or with weighted A* for a suboptimal) path to the goal state*



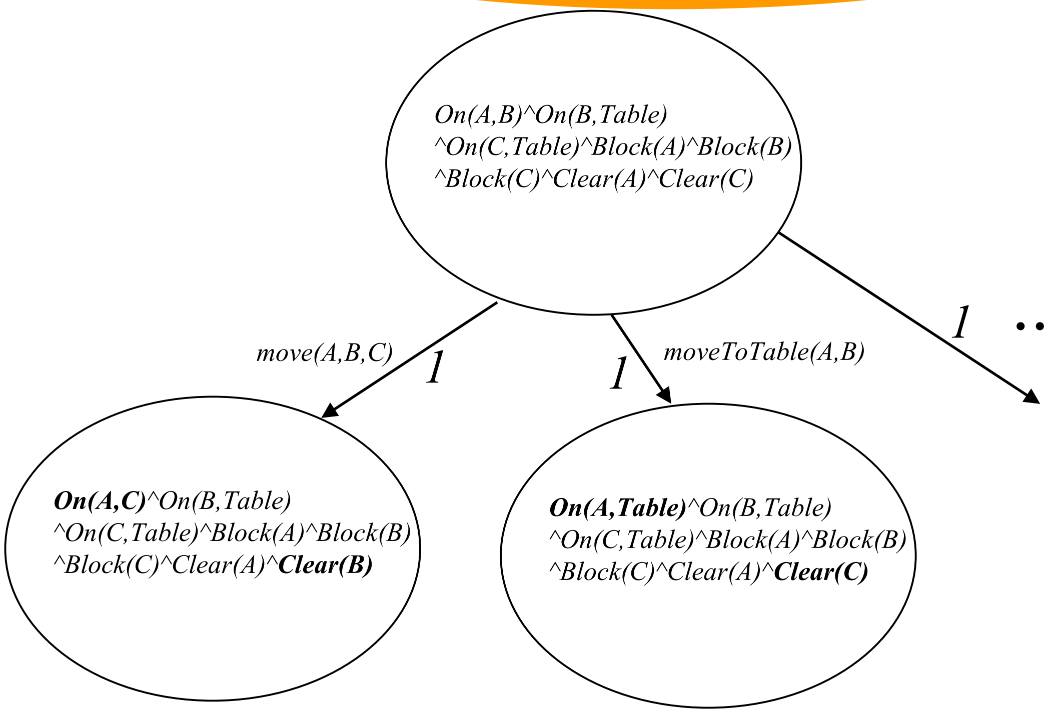
Planning via Graph Search

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



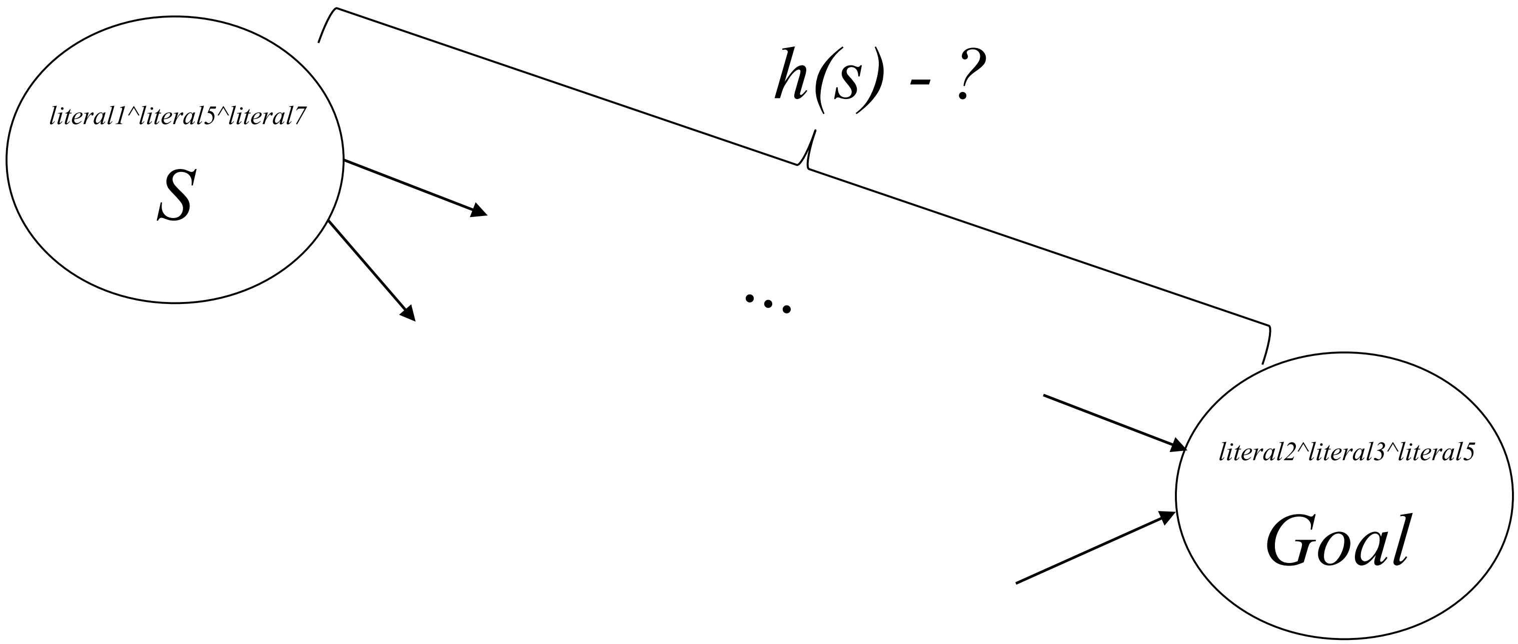


Assign edgecosts and search with A for a least-cost (or with weighted A* for a suboptimal) path to the goal state*



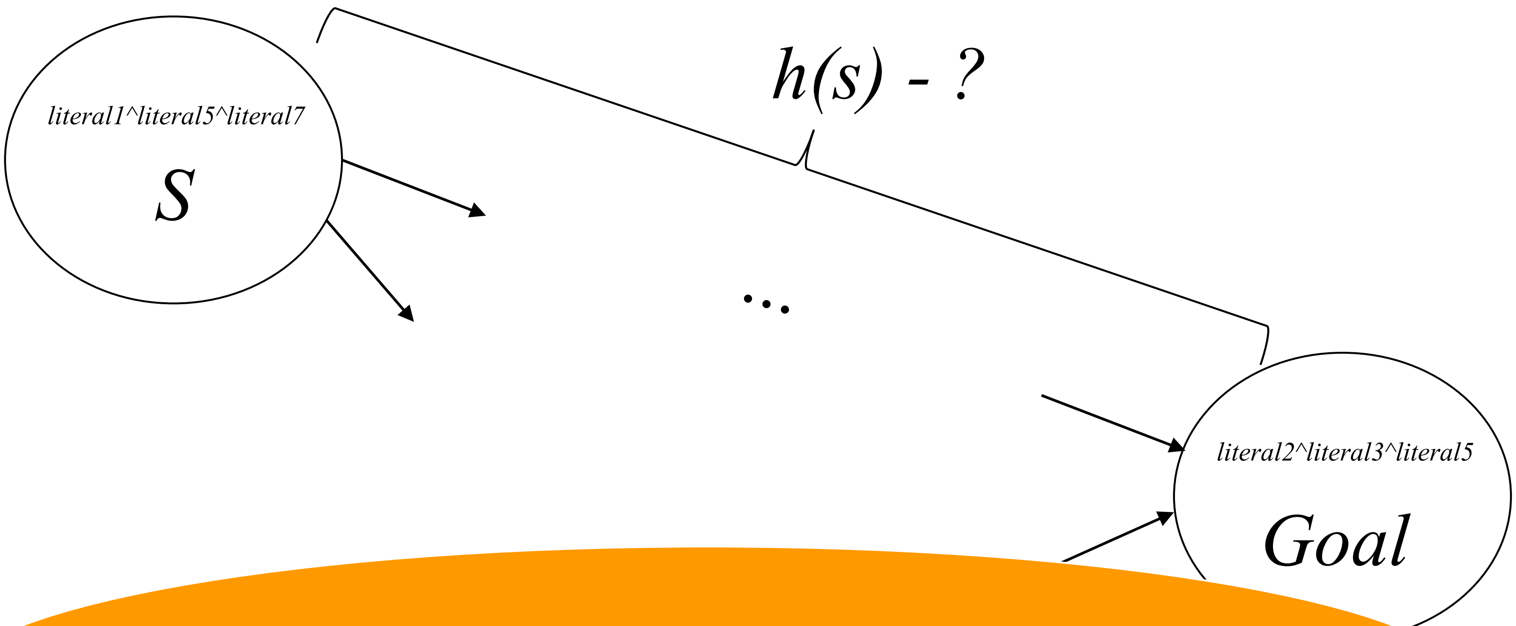
Planning via Graph Search

- Computing heuristics



Planning via Graph Search

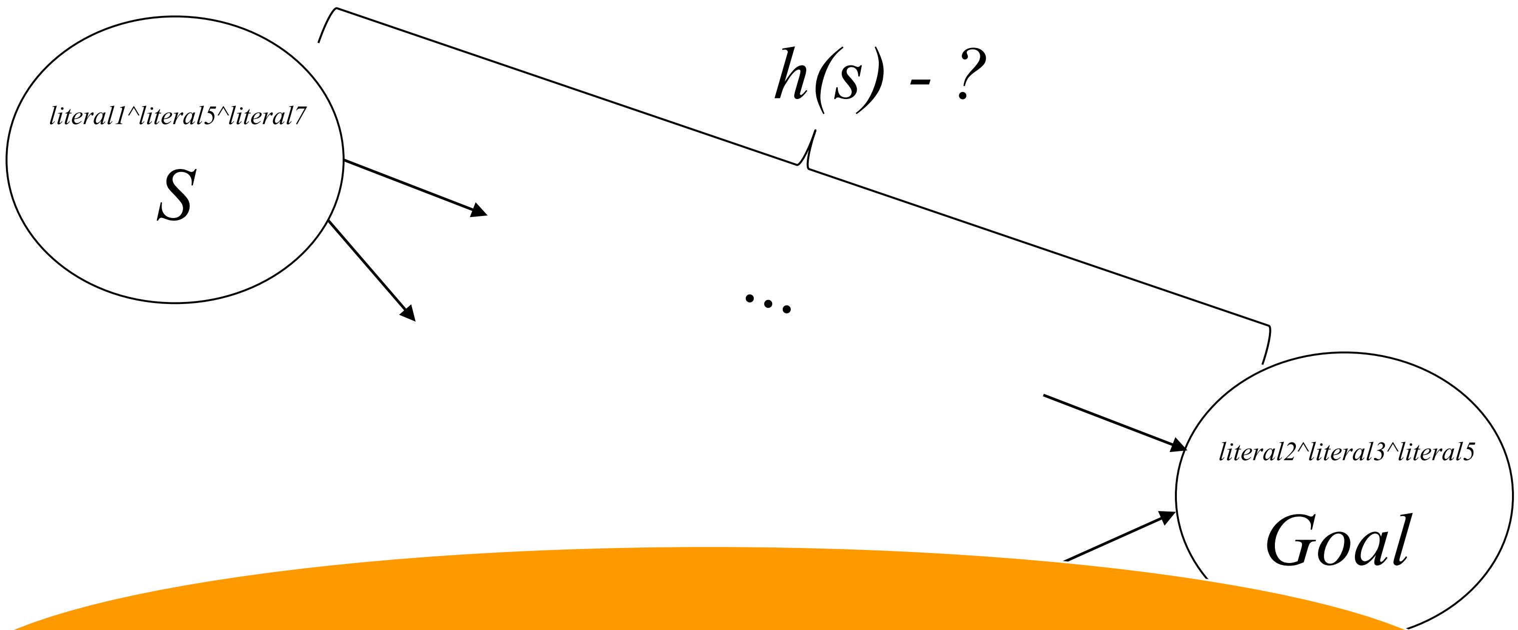
- Computing heuristics



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



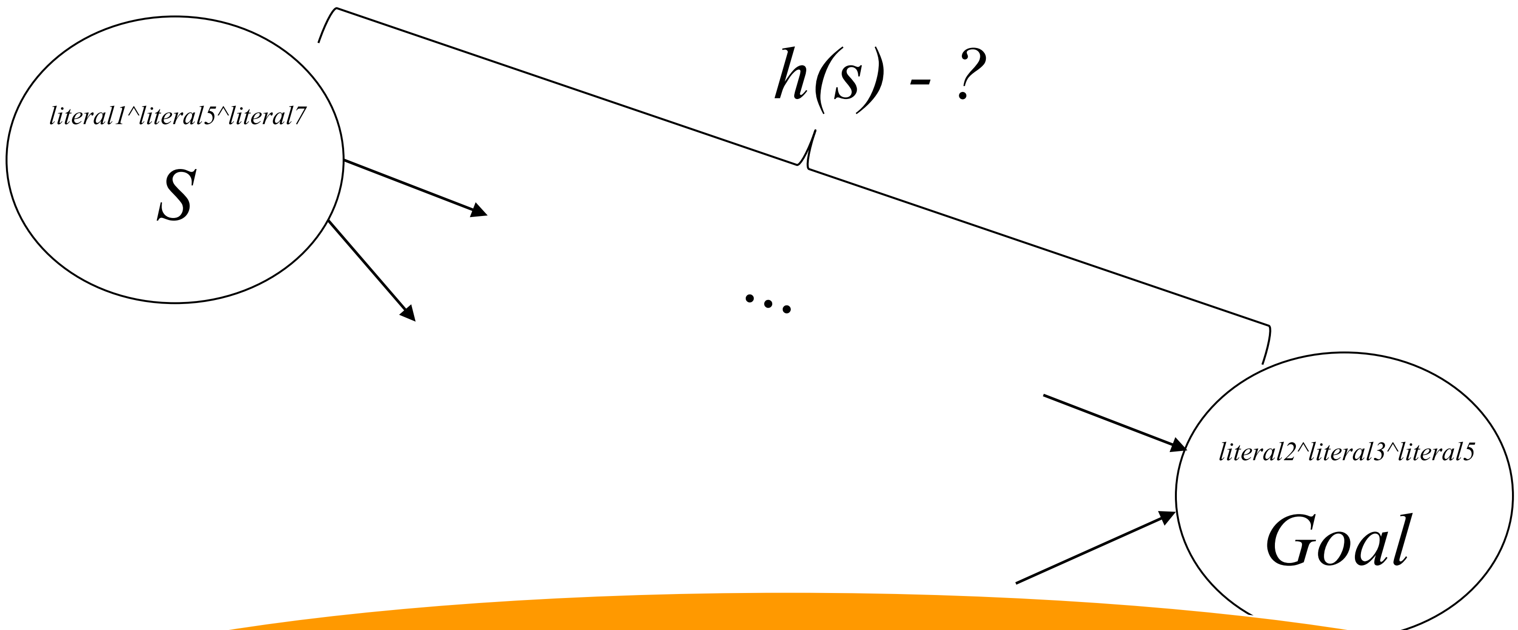
Option 1: h(s) = # of literals that are NOT yet satisfied
i.e., h(s) = # of literals l_i such that l_i(s)=false and l_i(goal) = true

Is this heuristic function admissible?

Can we still use it? What do we sacrifice?

Planning via Graph Search

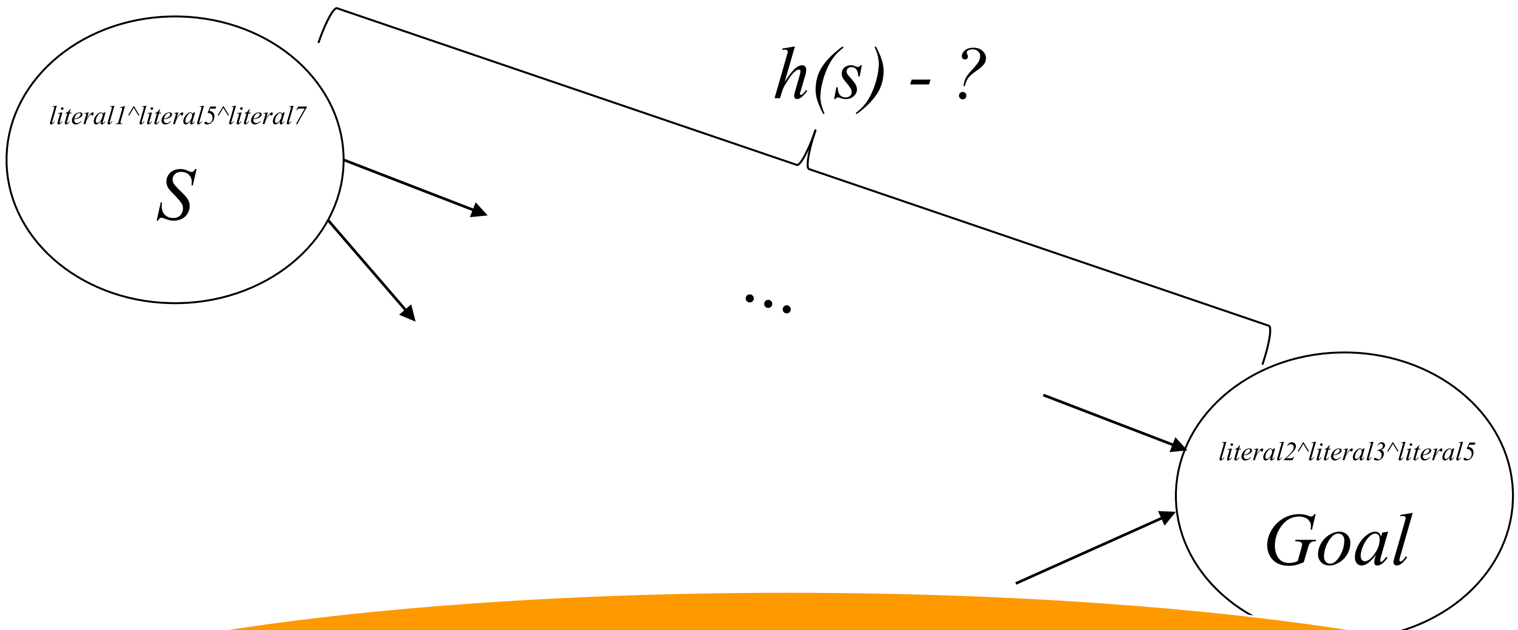
- Computing heuristics



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

Planning via Graph Search

- Computing heuristics



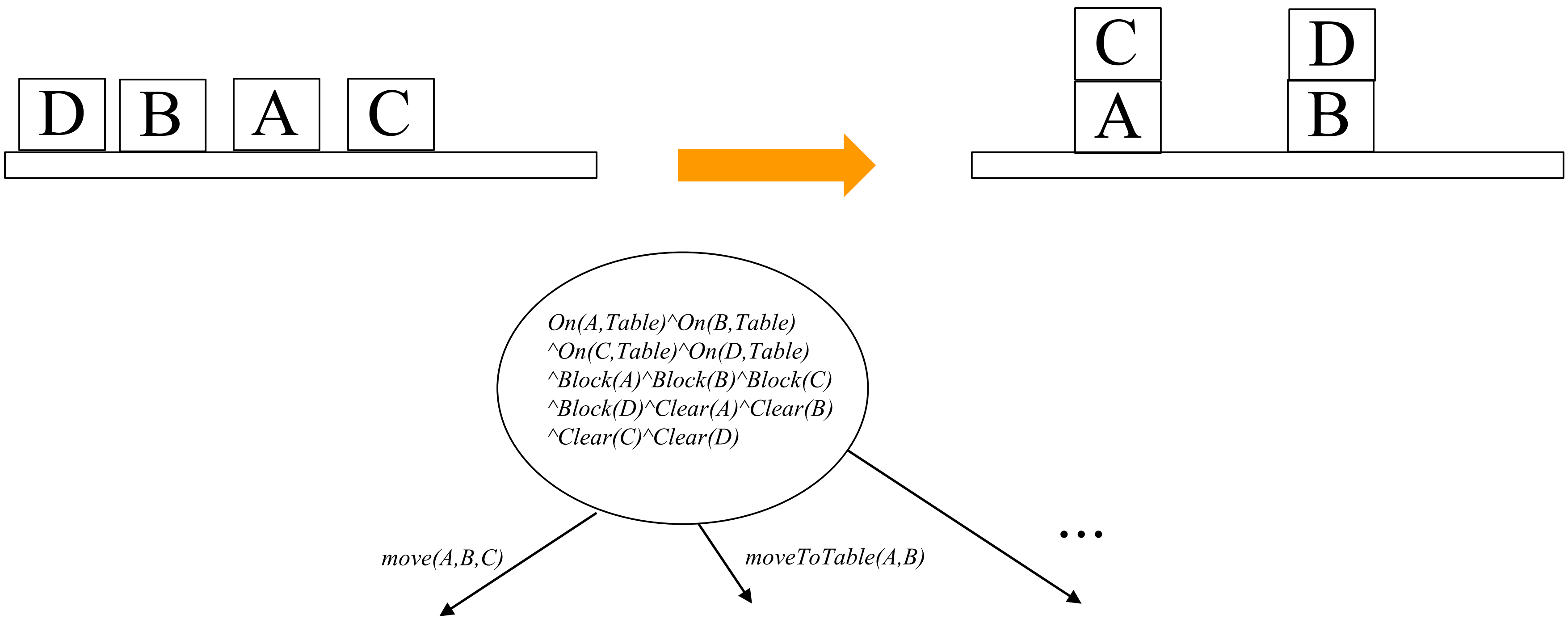
Option 2: compute heuristics using a ***relaxed*** *(simpler) problem*
Common relaxation: assume actions don't have any *<u>negative</u>* *effects*
(called empty-delete-list heuristics)

Any downsides?

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

Planning via Graph Search

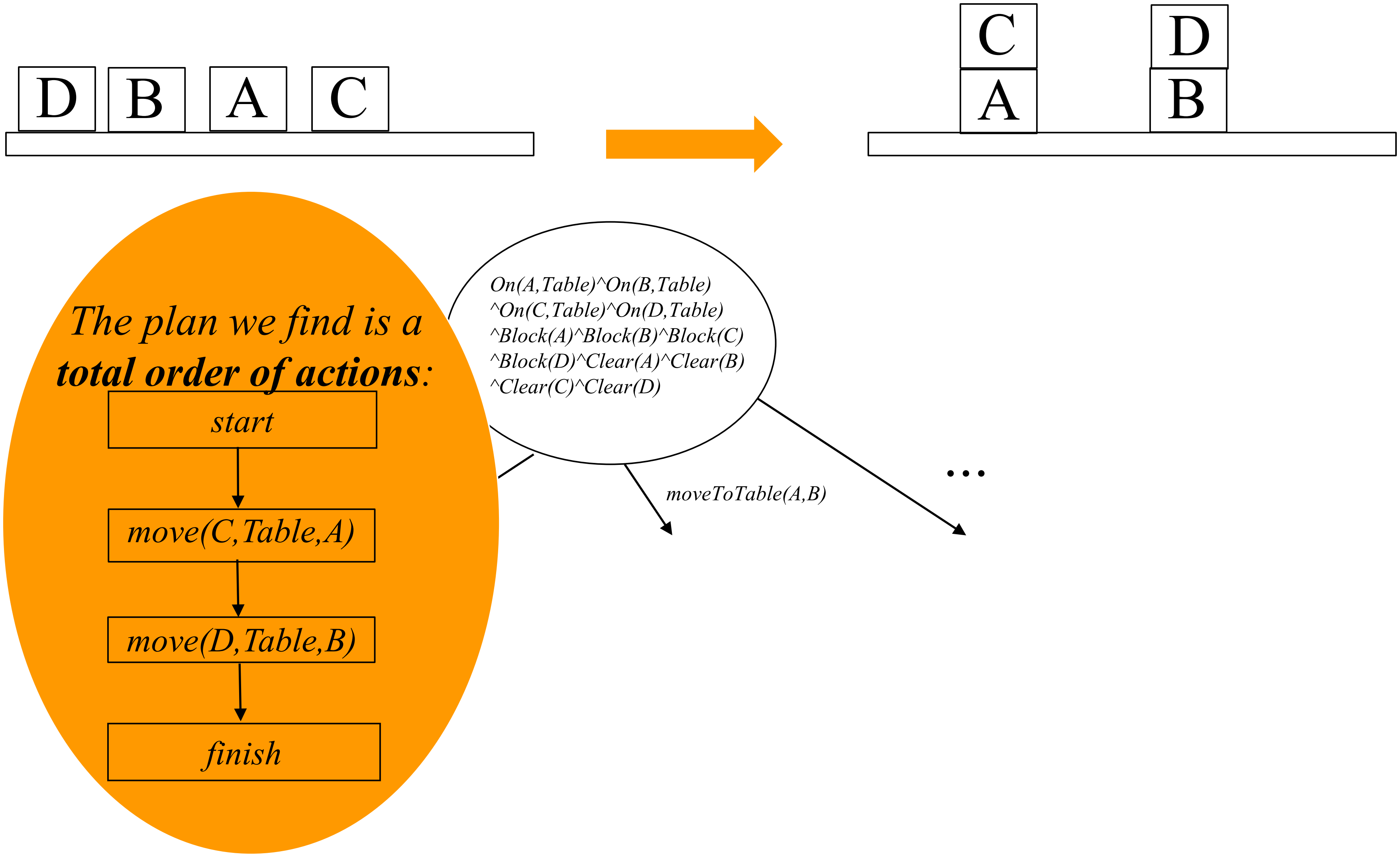
- Challenges in graph search formulation



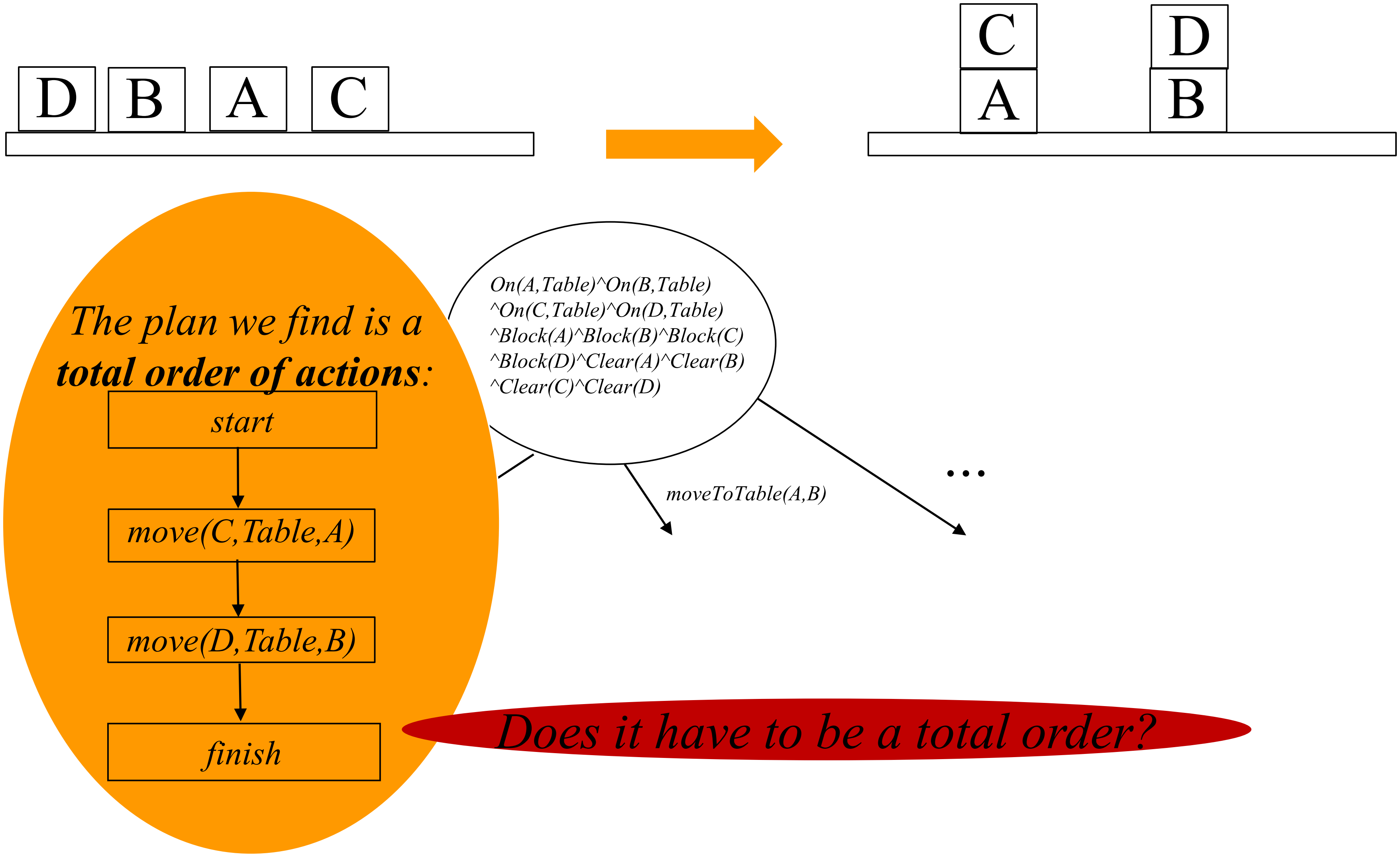
How many potential successors?

Planning via Graph Search

- Challenges in graph search formulation

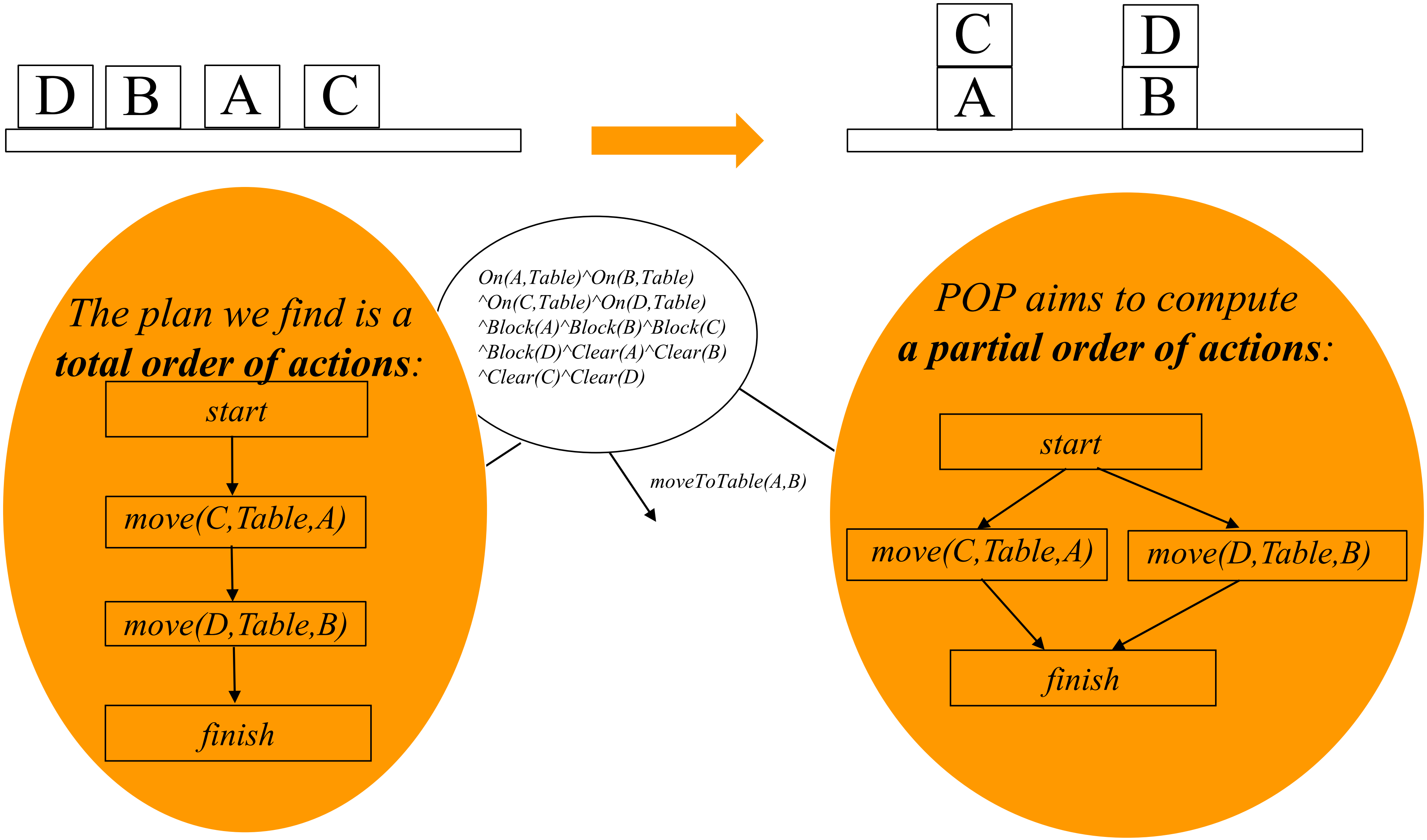


- Challenges in graph search formulation



Partial-Order Planning (POP)

- Total vs. partial ordering of actions

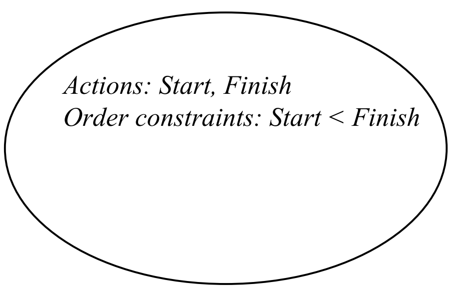


Partial-Order Planning (POP)

- 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 $A \overset{p}{\Rightarrow} B$ (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 <u>at some point</u> 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 $A \overset{p}{=>} B$ (action A achieves precondition *p* required by action B)



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 <u>at some point</u> before action B). No cycles allowed (i.e., A<B and B<A is a cycle and makes such state invalid)
    - Se… dition *p*

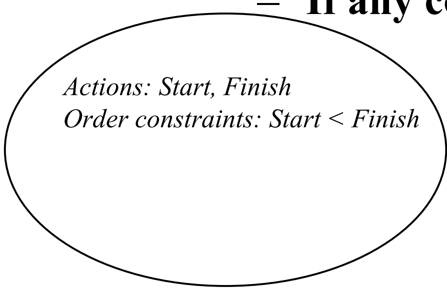
Start action has: no preconditions; effect=the literals in the actual start state
Finish action has: preconditions=the literals in the actual goal state; no effect

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 $A \overset{p}{\Longrightarrow} B$ 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 $D \overset{p'}{\Longrightarrow} F$, then $A<D$ or $F<A$ added
      - **If any constraint cycle is introduced, then *S'* is an invalid successor**



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 $A \overset{p}{\Rightarrow} B$ 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 $D \overset{p'}{\Rightarrow} F$, then *A<D* or *F<A* added
      - **If any constraint cycle is introduced, then *S'* is an invalid successor**

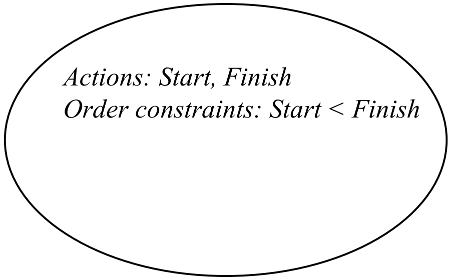
Actions: Start, Finish
Order constraints: Start < Finish

Start state

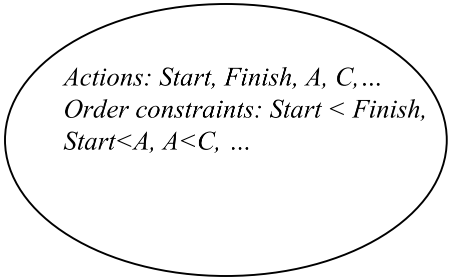
This gives us an implicit graph
that is typically searched by Depth-First Search
for any feasible solution to the goal state

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 symbolic planning can be represented as a graph search and solved with heuristic searches (A*, weighted A*, etc.)

- Few ways for how domain-independent heuristics can be computed automatically

- Overall understanding of what Partial-order Planning is